TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

**JARMO HAKALA**

# OBJECT RECOGNITION FOR MARITIME APPLICATION USING DEEP NEURAL NETWORKS

Master of Science Thesis

Examiners: Professor Heikki Huttunen and D.Sc. Mika Hyvönen

The examiners and topic of the thesis were approved on 30th May 2018

# ABSTRACT

**JARMO HAKALA**: Object Recognition for Maritime Application Using Deep Neural Networks
Tampere University of Technology
Master of Science Thesis, 55 pages
November 2018
Master's Degree Programme in Information Technology
Major: Data Engineering and Machine Learning
Examiners: Professor Heikki Huttunen and D.Sc. Mika Hyvönen

Keywords: Machine Learning, Object Recognition, Deep Learning, Convolutional Neural Network

The aim of this thesis was to study object recognition with the state of the art methods in order to evaluate their potential for the sector of autonomous maritime logistics. In autonomous maritime transportation, object localization and recognition is crucial for safe and efficient traffic flow. In this study, object recognition was studied by training deep convolutional neural networks for image classification and by evaluating their classification and computational performance.

In machine learning, a classification algorithm is trained with supervised learning with a dataset of input-output examples. Object recognition is a classification task where objects are classified from images. In deep learning, deep neural networks with multiple layers learn hierarchical representations of the data. For training, they require more computation and data than traditional machine learning methods. In the past years, more and more data has become available, and the computation capacity has increased dramatically. Therefore, deep neural networks have outperformed traditional machine learning algorithms in many tasks, such as object recognition. The best results in object recognition are achieved using deep convolutional neural networks.

In the experiments, deep convolutional neural networks were trained for image classification with Rolls-Royce Maritime Image (RRMI) dataset. Small-CNN architecture was generated and trained with random hyperparameter search approach using random weight initialization whereas VGG16, ResNet50 and MobileNet architectures were trained with transfer learning. The classification and computational performances of the models were measured. Transfer learning approach proved to improve classification performance. The VGG16 achieved the best accuracy of 84.0% for the dataset. The best average class accuracy of 78.4% was achieved with the ResNet50. The computational performance of the models was evaluated by measuring the time required for image classification with a CPU and GPU in order to evaluate their potential for a real-time object localization and recognition system. With the GPU, the models were much faster and performed in $3.6 - 16.0$ milliseconds.

# TIIVISTELMÄ

**JARMO HAKALA**: Hahmontunnistus merenkulkusovelluksessa syviä neuroverkkoja käyttäen
Tampereen teknillinen yliopisto
Diplomityö, 55 sivua
Marraskuu 2018
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Data Engineering and Machine Learning
Tarkastajat: Professori Heikki Huttunen ja TkT Mika Hyvönen

Avainsanat: Koneoppiminen, Hahmontunnistus, Syväoppiminen, Konvoluutioneuroverkko

Tämän diplomityön tarkoitus oli tutkia hahmontunnistusta tämän hetken huipputason menetelmien avulla ja arvioida niiden käyttöä autonomista laivaliikennettä varten. Hahmojen paikannus ja tunnistus ovat erittäin tärkeitä tekijöitä autonomisen laivaliikenteen turvallisuuden ja tehokkuuden vuoksi. Tässä tutkimuksessa hahmontunnistusta tutkittiin kouluttamalla syviä konvoluutioneuroverkkoja kuvantunnistukseen ja arvioimalla niiden luokittelu- ja suorituskykyä.

Koneoppimisessa luokittelualgoritmi koulutetaan ohjatun oppimisen avulla aineistolla, joka koostuu sisään- ja ulostuloarvoista. Hahmontunnistus on luokittelutehtävä, jossa hahmoja tunnistetaan kuvista. Syväoppimisessa monikerroksiset syvät neuroverkot oppivat esittämään datan hierarkisesti. Koulutusta varten syvät neuroverkot tarvitsevat enemmän dataa ja laskentatehoa kuin perinteiset koneoppimisen menetelmät. Viime vuosina datan määrä ja laskentakapasiteetti ovat kasvaneet erittäin paljon. Tämän vuoksi syvät neuroverkot ovat menestyneet perinteisiä koneoppimisen algoritmeja paremmin monissa tehtävissä, kuten hahmontunnistuksessa. Nykypäivänä parhaat tulokset hahmontunnistuksessa saavutetaan käyttäen syviä konvoluutioneuroverkkoja.

Kokeissa syviä neuroverkkoja koulutettiin kuvantunnistukseen Rolls-Royce Maritime Image (RRMI) dataa käyttäen. Small-CNN arkkitehtuuri luotiin ja koulutettiin satunnaisparametri ja -alustusarvojen avulla, kun taas VGG16, ResNet50 ja MobileNet arkkitehtuurit koulutettiin siirto-oppimisen avulla. Mallien tarkkuutta ja suorituskykyä mitattiin. Siirto-oppimisen todettiin parantavan tarkkuutta. VGG16 malli saavutti parhaan 84,0%:n tarkkuuden. Parhaan keskimääräisen luokkatarkkuuden saavutti ResNet50 malli 78,4%:n tarkkuudella. Mallien suorituskykyä arvioitiin mittaamalla aikaa, joka tarvittiin kuvantunnistukseen CPU:n ja GPU:n avulla, jotta mallien potentiaalia reaaliaikaisessa hahmonpaikannus ja -tunnistus -sovelluksessa voitaisiin arvioida. GPU:n avulla mallit olivat paljon nopeampia ja suoriutuivat $3{,}6 - 16{,}0$ millisekunnissa.

# PREFACE

This thesis was done in co-operation with Rolls-Royce as a part of DIMECC's D4V project. In the thesis, object recognition using deep neural networks was studied in order to evaluate their potential for a maritime application.

First and foremost, I would like to thank my supervisor Mika Hyvönen for the positive feedback during the work. I am equally grateful to my other supervisor Professor Heikki Huttunen for the valuable discussions on the subject. I am also thankful for all the comments I have received from Anssi Lappalainen, Eero Lehtonen and Jussi Poikonen during the meetings. I would like to extend my thankfulness to everyone else who participated in the project.

In addition, I appreciate all the help I have received from my friends during the thesis. I would also like to express my gratitude to my relatives, brother, mother and passed away father for all the encouragement during my studies and being there for me. Finally, a sincere thank you belongs to Pauliina for supporting me, especially during the last months of the thesis.

Tampere, on 24th October 2018

Jarmo Hakala

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ABBREVIATIONS

| | |
|---|---|
| AI | Artificial Intelligence |
| AUC | Area Under Curve |
| CNN | Convolutional Neural Network |
| COLREGS | International Regulations for Preventing Collisions at Sea |
| CPU | Central Processing Unit |
| DL | Deep Learning |
| DNN | Deep Neural Network |
| D4V | Design for Value program |
| FC | Fully connected layer |
| FN | False Negative |
| FNN | Feed-forward Neural Network |
| FP | False Positive |
| FPR | False Positive Rate |
| GPU | Graphics Processing Unit |
| ILSVRC | ImageNet Large Scale Visual Recognition Competition |
| LiDAR | Light Detection and Ranging |
| MSE | Mean Squared Error |
| NN | Neural Network |
| IMO | International Maritime Organization |
| RAM | Random-Access Memory |
| RCC | Recurrent Neural Network |
| ReLU | Rectified Linear Unit |
| ROC Curve | Receiver Operating Characteristic Curve |
| SGD | Stochastic Gradient Descent |
| SVM | Support Vector Machine |
| TN | True Negative |
| TP | True Positive |
| TPR | True Positive Rate |
| XML | Extensible Markup Language |

# 1. INTRODUCTION

Artificial intelligence (AI) enables autonomous transportation, advanced health care and many more innovations [12]. But what is it? Oxford dictionary defines AI as follows: "The theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages" [36]. Machine learning (ML), in turn, is a field of science and engineering which uses algorithms to extract patterns from data. AI and ML are linked very closely to each other. While AI can act and make decisions on its own, machine learning can be used to solve different tasks of an AI system. Machine learning tasks include, for example, anomaly detection, machine translation and object recognition. Much of the recent development in AI is closely related to the latest development with ML and deep learning (DL) in particular.

Most of the recent success in ML has been achieved using neural networks (NNs) which is an algorithm used for supervised tasks like classification. NNs consist of layers of neurons. Shallow NNs with a few layers have already been around since the 1960s, though, their use was limited for simple problems [43]. However, this has changed. With more data and computation available nowadays, deep learning with deep neural networks (DNNs) outperform traditional machine learning algorithms in several tasks [43][31]. One of these tasks is object recognition where images of different objects are classified into discrete categories. Object recognition is the main focus of the thesis.

In object recognition, DNNs have become the state of the art method [43]. ImageNet Large Scale Visual Recognition Competition (ILSVRC) is a competition where the best algorithms in the field of object detection and recognition are evaluated. Year 2012 marked a breakthrough for object recognition and deep learning in general. In that year's competition, *Krizhevsky et al.* were able to improve the benchmark classification accuracy by a great margin using a deep learning approach with deep convolutional neural network (CNN) [30]. Since then, CNNs have been applied extensively for object detection and recognition problems with great results [51][9][22][25].

This thesis was done as a part of the DIMECC's project Design for Value (D4V). The goal for D4V is described by DIMECC as follows: *"The DIMECC Design for Value (D4V) program focuses on door-to-door supply chain which is under digital disruptions and is rapidly changing towards an ecosystem of fully autonomous system-of-systems. Program work is conducted in two interrelated areas: autonomous maritime logistics and manufacturing use cases." [10]* Safe and efficient traffic flow are important issues for maritime logistics sector. The regulations and laws of the sector concerning these issues are regulated by the global standard-setting authority International Maritime Organization

(IMO) which is a specialized agency of the United Nations. One of these international regulations is International Regulations for Preventing Collisions at Sea (COLREGS). COLREGS contains many regulations and guidelines on how to act in various situations at the sea, for example, when coming across another vessel. [26] For this reason, object localization and recognition is crucial for maritime logistics, especially for autonomous maritime logistics.

One part of the D4V project focuses on the technological aspects of autonomous maritime logistics, in particular object localization and recognition. In this subproject, the goal was to build a real-time multi-sensor system for object localization and recognition using synchronized Light Detection and Ranging (LiDAR) and camera sensors. The proposed system would use LiDAR for object localization by generating regions of interest of possible objects. These regions would then be extracted as images with the synchronized camera feed and input to a model for recognition. Similar approaches for localization and recognition for autonomous vehicles have been researched in [7][35].

This thesis focuses on the object recognition part of the previously described system by researching and experimenting with the state of the art methods used for object recognition. The thesis also provides the theoretical background for machine learning and deep learning. We focus on neural networks and in particular convolutional neural networks. In addition, we present practical methods for training deep neural networks. For experimentation, we use Rolls-Royce Maritime Image (RRMI) dataset which consists of images in maritime environment. In the experiments deep CNNs are trained for image classification with different training approaches and their classification performance for the dataset are evaluated. For a real-time object recognition application, rapidity of classification is important. For this reason, computational performance of the models is evaluated by measuring the time required for image classification. After experimentation, we discuss and compare the results and overall implementation with related studies. In addition, we consider limitations of the study and present ideas for future research.

The rest of the thesis is organized as follows. Chapter 2 provides a theoretical background for the thesis. It provides an overview on machine learning and neural networks. Practical methods for training deep neural network are discussed. In addition, the chapter deals with methods for classier evaluation. The end of the chapter presents studies related to object recognition. In Chapter 3, research data and methods used for thesis experimentation are described. In Chapter 4, the experimental results are presented and evaluated. In Chapter 5, the thesis experimentation and its limitations are discussed and compared with related studies. In addition, suggestions for future work are provided. Conclusions of the thesis are presented in Chapter 6.

# 2. THEORETICAL BACKGROUND

This chapter provides a theoretical background for the research of the thesis. First, a short overview on machine learning is provided, concentrating particularly on supervised learning and deep learning. We continue with neural networks (a method for supervised learning) by discussing their structure, training and regularization methods. After neural networks, we handle convolutional neural networks which are the state of the art method in object recognition. In addition, we present practical methods for training deep neural networks and introduce metrics of a classifier. The chapter is concluded with studies related to thesis research.

## 2.1 Machine Learning

Machine learning aims to make computer programs which are able to learn and improve progressively. Machine learning has had a great impact for the recent success in AI. One of the first times when the term machine learning was mentioned was in 1959 when a computer program was trained to play checkers with machine learning [42]. There are several different ways of describing what machine learning actually is. One of the most commonly cited description is provided by Mitchell: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [34, p.2]. The purpose of a machine learning process is to train an algorithm to perform better in selected task(s) measured with respect to a performance metric [17, pp. 97-98]. When algorithms are executed with data, they "learn" from the process [17, p.97]. In this way the algorithms perform better in the following executions.

Depending on the nature of a problem, there are three distinct approaches for training a machine learning algorithm, namely **supervised learning**, **unsupervised learning** and **reinforcement learning** [3, p.3][17, pp. 102-104]. In supervised learning, an algorithm is trained with a dataset of input-output examples. In training, the algorithm learns to map inputs to specific outputs by observing patterns in the data. Supervised learning can be used, for example, classification and regression tasks. In unsupervised learning, the dataset consists of input examples without their target output. Unsupervised learning algorithms can be used to measure similarity or dissimilarity among the inputs by observing their patterns. Unsupervised learning tasks include, for example, cluster analysis. In reinforcement learning, an algorithm does not have a finite dataset for training but it must extract data by sensing and interacting with its environment. Some of the actions are rewarded and some are punished. In this way, the algorithm reinforces the behavior which results in rewards and avoids actions which lead to punishments. Next, we focus in

supervised learning which is the method for training a classifier.

### 2.1.1 Supervised Learning

Supervised learning is the most common method for training a machine learning algorithm [31]. There are several different algorithms which can be used for supervised learning e.g. neural networks, decisions trees and support vector machines [29]. Some algorithms perform better in some problems while other algorithms perform better in other problems. Classification and regression tasks are solved with supervised learning. For training, a supervised algorithm requires a dataset of input-output examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ where $\mathbf{x}_n \in \mathbb{R}^i$. In regression tasks, outputs are continuous $y_n \in \mathbb{R}$. For example, a regression problem could be, for example, to predict price of a house based on its size and location. For classification, outputs are discrete categories $y_n \in \{1, 2, \ldots, k\}$ where $k$ is the number of different categories being classified. For example, a classification problem could be to predict if the house is over a certain threshold price.

Before supervised training, the dataset is split into **training set** and **test set**, shown in Figure 2.1. A simple way for the partition is to split most of the data for training and some of it for testing, for example 80% into training set and 20% into test set. The split is commonly done in stratified manner so that the sets contain the same distribution of different examples in them [14]. As the names imply, the training set is used to train a model, whereas the test set is used for testing the model with previously unseen examples. The ability of a model to predict previously unseen examples correctly is called **generalization**. When a machine learning problem is being solved, several models may be trained and tested with the data. If the test set is used several times to estimate generalization, it stops serving its purpose. A good generalization performance may have been achieved by chance. In this case, a separate **validation set** can be used. Validation set is a subset of the training set as shown in Figure 2.1. It can be used for initial generalization estimation. After training multiple models, the final generalization performance is estimated only with the model which achieved the best validation performance for the test set.

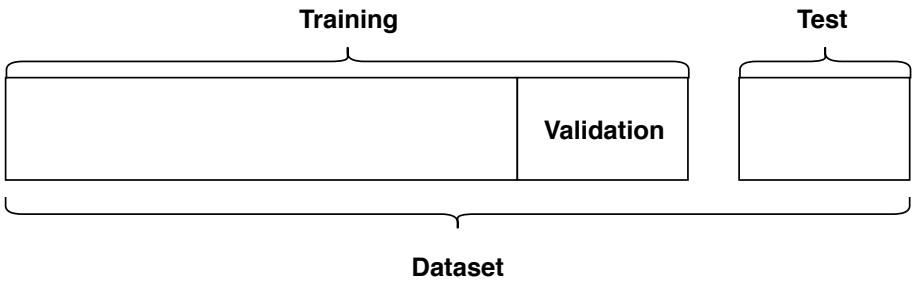



***Figure 2.1.*** *Subsets of a dataset used for training a machine learning model.*

A machine learning model is generated by fitting a function for the training set using a selected algorithm. Linear regression is simple supervised learning algorithm [17, p.105]. Linear regression is defined as

$$y = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}, \tag{2.1}$$

where each input value $x_i$ of input vector $\mathbf{x}$ is weighted with weight parameter $w_i$ of weight vector $\mathbf{w}$ to predict the scalar output $y$ of the regression model. In order to measure how well a model performs, an error function is required. Error function measures the difference between the expected output and model output. Mean squared error (MSE) is a simple error function machine learning where the error $E$ with a set of weights $\mathbf{w}$ is defined as

$$E(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} (f(\mathbf{x}_n, \mathbf{w}) - y_n)^2, \tag{2.2}$$

where the error is calculated for $N$ examples by comparing their model output with expected target output $y_n$ [17, p.105]. The smaller the error the better a model is fit for the training data. By adjusting of the weights of the model the error can be decreased. The fitness of a model can be affected by altering its capacity. The capacity can be affected, for example, by increasing the number of parameters in the model or by changing the learning algorithm itself. By increasing the capacity of a model, it is possible to perfectly fit a function and reach zero error for the training set. However, this is undesired because we want to train a generalized model for the problem. [17, pp.105-109]

The effect of model's capacity with respect to model training and generalization error is shown in Figure 2.2. If a model has a too small capacity, the model is **underfitting**. In this case, the model is unable to fit for the training data which leads to poor training and generalization performance. By increasing the capacity of a model, it is possible to reach smaller and smaller training error. At some point, generalization error starts to increase. After this point, models are **overfitting**. In overfitting, a model fits too well for the training set. In training, we want to train a model with an optimal capacity in order to achieve the lowest generalization error for the model. This point is shown as the red line in Figure 2.2. [17, pp. 109-117]

Overfitting is a common problem in machine learning. However, there are practical ways to avoid it. If an algorithm with a too great capacity is trained with a too small training set, the model is likely to overfit. When the training set size is increased, overfitting becomes a less severe problem. A greater training set allows to use a more complex model with greater capacity without having to worry about overfitting. [3, p.9]

Another way to avoid overfitting is regularization. With regularization, the purpose is to restrict the training of an algorithm in order to prevent it from perfectly fitting the training data. This way, models with greater capacity can be trained for less complex tasks without having to worry about overfitting. [17, p.118]

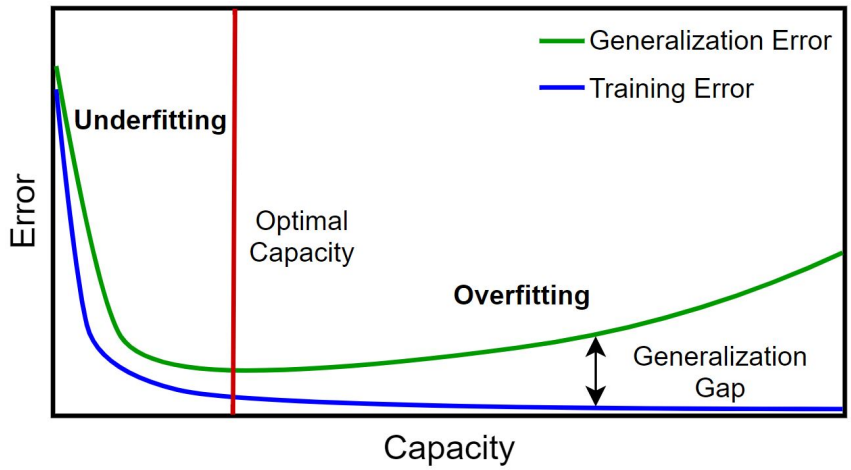



***Figure 2.2.*** *Model capacity with respect to training and generalization error (Adapted from [17, p.113]).*

### 2.1.2 Deep Learning

Traditional machine learning methods struggle to perform as the dimensionality of the input data increases [31]. For example, in image classification input images may contain thousands of pixel values. This problem is called the curse of dimensionality [17, p.152]. To overcome this problem, traditional methods require manual feature engineering of raw data in order to decrease the input dimensions [31][17, pp. 3]. However, in practice, engineering these features from a mass of data is hard without specific domain expertise [31][17, pp. 3]. This is where deep learning steps in.

Deep learning solves the curse of dimensionality by learning hierarchical representations of the data with complex models of multiple layers. This is achieved with representation learning. In representation learning, no feature engineering is used. Instead, a model learns by itself which features of the mass of data are relevant for the task. When a deep learning model is trained, the model learns abstract representations of the data by observing the data patterns. The first layers of a deep learning model learns more simple features of the data, whereas the last layers learn more specific features of the data.

Why has deep learning become popular? Deep learning models require considerable amounts of data for training [17, p.10]. Nowadays, during the era of big data, increasingly more data is being generated. This is one reason for the change. The more there is data, the better it is for model generalization. Deep learning models are also much greater in size than traditional machine learning models. The greater size increases their modeling capacity and enables them to solve very complex problems but increases their computational burden [17, p.21]. Because of the computational burden, use of deep learning was impossible before. This has changed. Compared to computation capacity in 1990s, the computation capacity of a modern graphics processing units is over a million times faster [43]. In addition to the computation development, the technologies and software used for deep learning have made great progress [17, p.21].

Most of deep learning, as machine learning in general, is performed using supervised learning [31]. Deep learning models have been studied to outperform traditional machine learning methods in many different tasks [31]. Some of these tasks include speech recognition, object recognition and object detection. In practice, deep learning is almost always performed using deep neural networks. In the next section, we provide background for neural networks.

## 2.2 Neural Network

Neural network is a machine learning algorithm which is used for supervised learning tasks. Neural networks originate from 1950s [3, p.226]. Originally, the inspiration of NNs was the biological study of the brain and its billions of neurons which process and deliver information in a very complex network. [34, pp.81-82] The following sections deal with structure, training and regularization methods of a neural network.

### 2.2.1 Network Structure

A neural network is built of connected neurons. In Figure 2.3, a model of a single neuron is shown. In the following equations, this neuron is denoted as $j$. A neuron takes one to $i$ input values as inputs, which are summed, activated and then output. In the figure, the neuron's inputs are shown as nodes on the left-hand side of the figure. These are the input features of the neuron. The inputs are connected to neuron $j$ through connections with weight parameters. The weight parameters can be written as a vector. When the input sum is calculated, each weight parameter weigh the specific input connection. In addition to the normal inputs, every neuron has a constant input $x_0$ which equals 1. The weight of this connection is $w_{0j}$. This is the so-called bias parameter $b$. Together, all the input values of the neuron can be described as vector $\mathbf{x} = \begin{bmatrix} x_0 & x_1 & x_2 & \dots & x_i \end{bmatrix}^T$ and the parameters which weigh the values as vector $\mathbf{w} = \begin{bmatrix} w_{0j} & w_{1j} & w_{2j} & \dots & w_{ij} \end{bmatrix}^T$. The neuron combines the inputs with the weights and sums the values. The summation can be written with or without the bias parameter. The sum $z_j$ of the neuron is then transformed with a selected nonlinear **activation function** $h(\cdot)$ which produces the neuron output: [34, pp. 86-87][41, pp.727-728][3, pp. 227-228]

$$y_j = h(z_j) = h(\sum_{i=0}^{I} w_{ij}x_i) = h(\sum_{i=1}^{I} w_{ij}x_i + b). \tag{2.3}$$

A neural network is formed by stacking neurons into layers. Depending on the directions of the connections, neural network structures are divided into two types, namely feed-forward neural networks (FNN) and recurrent neural networks (RNN). In a FNN, the output connections of neurons are always connected to the following layers and the information always flows towards the outputs. Given an input, a FNN always predicts the same output

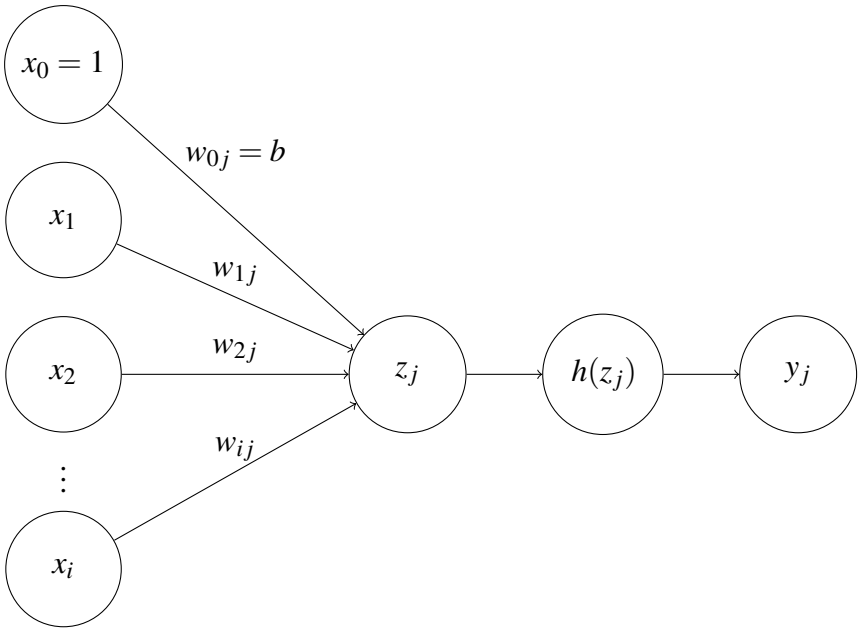



***Figure 2.3.*** *Model of a single neuron.*

value for the input. Contrary to a FNN, a RNN has feedback connections. The feedback connections are outputs of neurons which can be connected as inputs to preceding layers in a RNN. These feedback connections can change the internal state of a RNN and act as network "memory". Given an input, an RNN can output a different output depending on its previous inputs. [41, p.729] In the thesis, we focus on FNNs, in particular convolutional neural networks (CNNs) which is a type of a FNN. CNNs are handled later in the thesis in Section 2.3.

An example of a FNN is shown in Figure 2.4. The network consists of an input layer with three inputs, one hidden layers and an output layer with three outputs. A neural network can consists of any number of hidden layers but it always has an input and an output layer. The network shown in the figure is a fully connected network. Thus, every output of a preceding layer is connected to every neuron in its following layer. In the research, fully connected (FC) layers are sometimes called dense layers. [41, p.729]

The size and structure of a network have a great effect for its computational performance and capacity. Size of a network is reported with its total number of parameters. In a fully-connected network, all parameters are used to calculate the output. By increasing the number of parameters by adding layers or increasing the size (width) of single layers, the capacity of the network increases and it becomes capable to approximate more complex functions. However, this increases the size of the network which increases its computational burden because more operations are required to calculate the output. As a solution, the computational burden of a network can be reduced by using sparse connections and parameter sharing interactions. For example, CNNs utilize these interactions. [3, pp. 227-231][17, pp. 329-331]

For demonstration, the number of parameters in the network in Figure 2.4 is calculated.

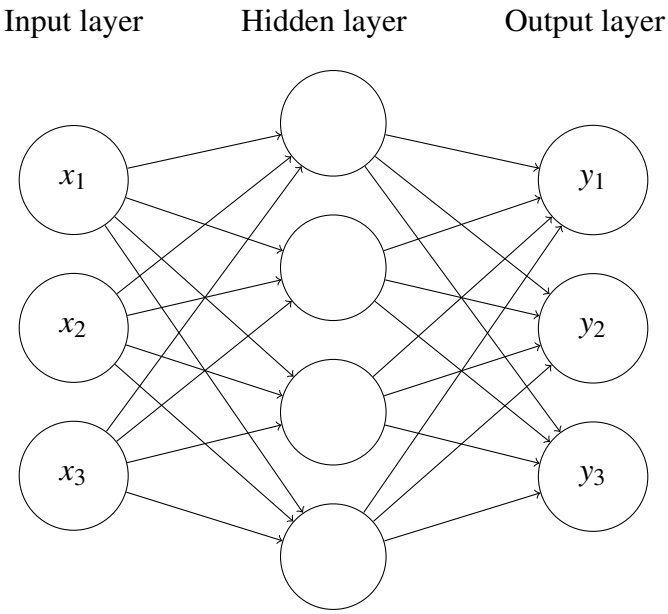



***Figure 2.4.*** *A fully connected three-layer FNN with three inputs and three outputs.*

The first hidden layer has $3 \times 4$ weights in addition to 4 bias parameters (bias connections are not shown in the figure for clarity). The output layer has $4 \times 3 + 3$ parameters. In total, the network consists of 31 parameters. An output of a network is calculated by propagating input values forward in the network layer by layer until the output layer prediction. In the example network in Figure 2.4, the input of the network consists of three features. Given an example, the network calculates three predictions, namely $y_1$, $y_2$ and $y_3$. An output prediction of neuron $k = 1,2,3$ at the output layer is calculated as

$$y_k(\mathbf{x},\mathbf{w}) = h\left(\sum_{j=0}^{J=4} w_{jk}^{(2)} h\left(\sum_{i=0}^{I=3} w_{ij}^{(1)} x_i\right)\right). \tag{2.4}$$

In forward propagation, activation functions apply nonlinear transformations for the inputs. With the transformations, the neural networks are able to represent more complex functions. Also, the nonlinear activation functions make the network optimization nonconvex, which allow the networks to be trained with iterative training methods [17, p.173]. In addition, the choice of activation function affects the training performance of the network. Next, we present four activation functions which have been and are currently used for neural networks. For demonstration, these action functions are also presented in Figure 2.5.

The first model of the neuron was the perceptron, which introduced by McCulloch and Pitts in 1943 [34, p.123][43]. The activation function of the perceptron was a hard threshold function:

$$h(z) = \begin{cases} 1 & z \geq 0 \\ -1 & \text{otherwise.} \end{cases} \tag{2.5}$$

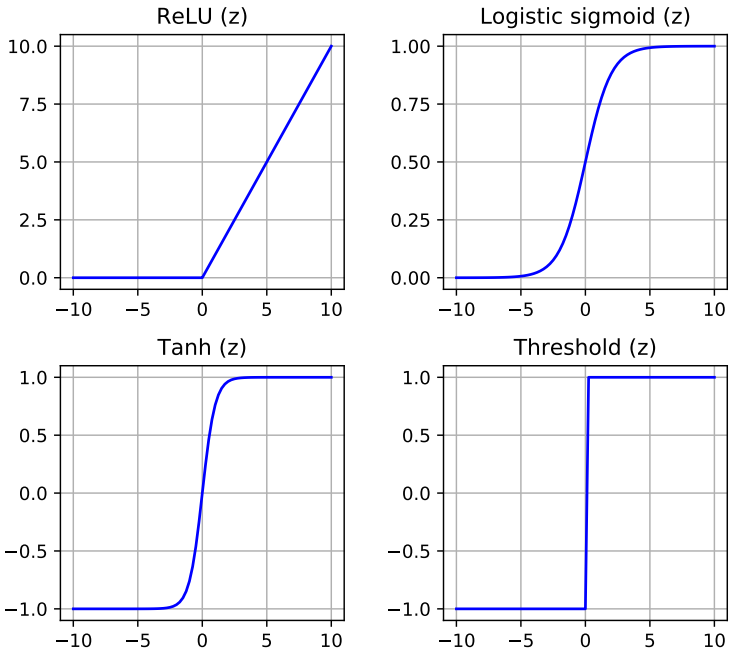



***Figure 2.5.*** *ReLU, Logistic sigmoid, Tanh and Threshold activation functions plotted for* $z \in [-10, 10]$.

In 1960s, perceptrons were used to form simple two-layer neural networks. These networks were unable to represent even very simple functions [41, p.22]. Due to technological constraints of the time, no effective algorithm was known in order to train deeper networks with increased model capacity [41, p.22]. In 1980s, the backpropagation algorithm was invented [41, p.22]. The backpropagation algorithm allows an efficient way to train multilayer networks. After backpropagation, network became deeper, and other activation functions were used to achieve better training performance [41, p.22][34, pp. 123-124] A commonly used activation function is the logistic sigmoid function:

$$h(z) = \frac{1}{1+e^{-z}}. \tag{2.6}$$

The logistic sigmoid function squashes its inputs to a range between 0 and 1. The output of sigmoid function can be interpreted as a probability. An other common activation function is the hyperbolic tangent function (tanh):

$$h(z) = \frac{e^{2z}-1}{e^{2z}+1}. \tag{2.7}$$

Tanh has a similar shape as logistic sigmoid but it squashes the inputs between $-1$ and 1. In 1990s, neural networks with sigmoid and tanh activation functions faced the vanishing gradient problem as networks became deeper [43]. When the input of these functions is either very small or very large, gradient of the function diminishes. This slows down the training of a neural network drastically. The magnitude of the problem is amplified as the

depth of a network increases [27]. The vanishing gradient problem can be prevented by using rectified linear unit (ReLU) activation:

$$h(z) = max(0, z). \tag{2.8}$$

Compared to sigmoid and tanh activation, ReLU does not suffer from the vanishing gradient problem. It's gradient is 1 with positive inputs and 0 otherwise. Nowadays, ReLU is the most commonly hidden layer activation function for deep neural networks [31]. It has been studied to decrease the time required for training deep neural networks [30][27].

### 2.2.2 Network Training

Neural networks are trained using supervised learning. The training is performed as an optimization task where the purpose is to minimize a selected error function by adjusting network parameters. The error function is also called a loss function. Before training, the size of input layer and output layer should be matched with dataset specifications. Moreover, output layer activation function should be matched with a corresponding loss function [3, p.236]. This subsection focuses on training a neural network for classification.

In a classification task, the dataset consists of input-output examples $(\mathbf{x}_n\ y_n)$ where $n = 1, 2, \ldots, N$. For training, we modify the target output $y_n$ to be a binary target vector $\mathbf{t}_n$. The binary target output vector is defined as $\mathbf{t}_n = \begin{bmatrix} t_{n1} & t_{n2} & \ldots & t_{nK} \end{bmatrix}$, where the subscripts $1, 2, \ldots, K$ are the classes of the dataset. Value $t_{nK}$ equals 1 if the example $\mathbf{x}_n$ belongs to class $k$ and is zero otherwise. For training, the size input layer is matched with dimensions of the inputs and the size of the output layer is matched with the number of classes $K$.

For classification tasks, MSE loss (Equation 2.2) can be used. However, using **cross entropy** loss function is preferred in order to achieve faster training and better model generalization [3, p.235]. Cross entropy loss is defined as

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\ ln\ y_k(\mathbf{x}_n, \mathbf{w}). \tag{2.9}$$

For binary classification tasks, the logistic sigmoid activation function can be used as the output layer activation function. With a single output neuron unit, the network's output is a prediction between zero and one. The sigmoid function can be generalized for a classification problem with $K > 2$ mutually exclusive classes. In this case, the generalized version of sigmoid function is used. This function is the **softmax** function:

$$y_k = \frac{exp(z_k)}{\sum_{j=1}^{K} exp(z_j)}. \tag{2.10}$$

A network with softmax function outputs a vector $\mathbf{y} = \begin{bmatrix} y_1 & y_2 & \dots & y_k \end{bmatrix}^T$ where $y_k$ is a probability between zero and one for the input belonging to class/category $k$. The values of the vector sum to one. For classification, the highest probability can chosen or a threshold value can be used. For a standard multiclass classification problem, softmax activation is used for output layer activation and the error of the network is calculated with cross entropy loss. [3, p.236]

The softmax activation is useful when estimating the overall performance of a classification model [33]. In multiclass classification task, the performance of a model can be measured with its accuracy. Accuracy is the proportion of correctly classified examples over all examples. Even though different models may have the same accuracy, one of the models may be more sure of its prediction. In Table 2.1, performance of two models is compared. Even though both of the models reach an accuracy of 75% with 4 examples, Model 1 is more sure in its predictions than Model 2 because the the output prediction values of the Model 1 are greater than the values of Model 2.

***Table 2.1.*** *Output comparison of two different neural network models with softmax.*

| **Output** | | | | | | | | **Target** | | | | **Result** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Model 1** | | | | **Model 2** | | | | | | | | **Model 1** | **Model 2** |
| **0.95** | 0.01 | 0.02 | 0.02 | **0.70** | 0.05 | 0.20 | 0.05 | 1 | 0 | 0 | 0 | Correct | Correct |
| **0.97** | 0.01 | 0.01 | 0.01 | **0.75** | 0.15 | 0.08 | 0.02 | 1 | 0 | 0 | 0 | Correct | Correct |
| 0.10 | 0.32 | **0.36** | 0.22 | 0.15 | 0.25 | **0.40** | 0.20 | 0 | 1 | 0 | 0 | Incorrect | Incorrect |
| 0.00 | 0.01 | 0.01 | **0.98** | 0.03 | 0.07 | 0.10 | **0.80** | 0 | 0 | 0 | 1 | Correct | Correct |

The loss function measures the error of a network with a set of weight parameters $\mathbf{w}$ by comparing the network output with the target output. For training, we want the find the weights which minimize the error of the network. This is achieved by using the gradient of the error function $\nabla E(\mathbf{w})$. As the gradient points to the direction of greatest increase in error, the negative of the gradient points to the direction the greatest decrease. The negative gradient is used to adjust the weights of a network towards a minimum of the error function until the gradient diminishes. We want to find a minimum point which results in a sufficiently small error $E(\mathbf{w})$. However, achieving a small error can be difficult. The bigger a network is in size, the more complex its weight space is. A simple error function with respect to its weights is plotted in Figure 2.6. The function has 3 minimums: the global minimum and 2 local minimums. Depending on the initial state of the weights (the red dots in the plot), the gradient can point towards a different minimum point. For deep neural networks, initialization of weights can have a great impact for finding a minimum with sufficiently small error. [3, pp. 236-237]

The way the gradient is used to minimize the loss is done using gradient descent [3, p.240]. In gradient descent, the weights are updated based on the error over the whole training set. Stochastic gradient descent (SGD) is modern approach of the gradient descent. SGD is one of the most used optimization methods for deep neural networks [17, p.290]. In

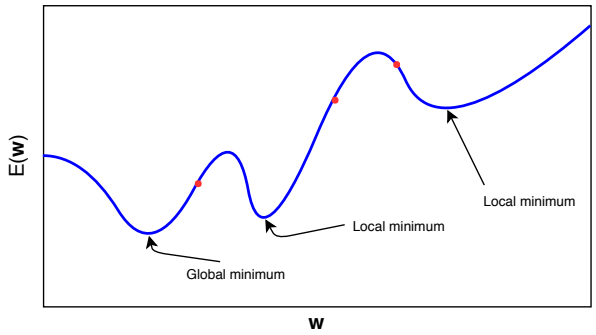



***Figure 2.6.*** *Example of an error space of a network with respect to its weights. The red dots in the plot display the error of the network with different set of weights. Depending on the initial weights, a different minimum may be reached in training.*

SGD, the weights of the network are updated in iterations by calculating the error $E_n$ for a selected number of independent examples. The weight update rule of SGD is defined as

$$\mathbf{w}^{i+1} = \mathbf{w}^i - \eta \nabla E_n(\mathbf{w}^i), \tag{2.11}$$

where $\mathbf{w}^i$ are the weights of the current iterations and $\eta > 0$ is the size of the adjustment which is also called the **learning rate**. The larger the learning rate, the larger the adjustment in weights is. When the weights are adjusted with the error over the whole training set of examples, the approach is batch learning. This is the approach of traditional gradient descent. As a downside, batch learning can be computationally heavy with a large dataset, and because of that even impractical [49]. To relieve the computational burden, SGD can be used with smaller training batches [17, p.275]. In on-line learning, the weights are adjusted based on the error of a single example. In mini-batch learning, the weights are adjusted based on multiple examples. *Wilson et al.* have studied the error and rate of convergence with on-line, mini-batch and batch learning approaches [49]. In their research, the training results of the approaches were observed to be similar. However, the time required for batch learning to reach the same level of accuracy was on average 20 times slower when compared with the other two approaches. Nowadays, a mini-batch size of 16 to 256 is preferred for training deep neural networks with SGD [17, p.276].

How to calculate the gradient of the error in order to efficiently adjust the weights of the network? This is achieved using the **backpropagation** algorithm. In this section, we provide a short explanation of backpropagation which is based on [3, p.244]. For the full mathematical description of the backpropagation algorithm, consider reviewing sources [3, p.241],[20, p.395] and [17, p.200].

Before backpropagation, forward propagation is performed. In forward propagation, examples are input to the network. During forward propagation, each of the neurons in the network form an activation output which is propagated forward along the weighted connections. The final predictions of the network are the outputs of the output layer neurons. Using the selected loss function, the output is compared with the target output

and the loss is measured. The output layer errors are back propagated to each neuron in the preceding layer according to the connections of the network. With these back propagated errors, the error on the preceding layer is calculated. Then, the weights between these layers are adjusted based on the partial derivatives and the learning rate. One can recursively apply the same method to adjust all of the weights of the network. [3, p.244]

The backpropagation is used to adjust the weights of the network mini-batch after a mini-batch based on their error. The network training can be performed for multiple iterations with the training set in order to reach an error which is small enough. One training iteration over the training set is called a training epoch [20, p.397].

As discussed in Chapter 2.1, we want to train a generalized model for a problem. During training, weights of the network are constantly adjusted so that training error decreases. If the training of the neural network is not regularized, the network can easily overfit [6]. An example of a training process with close to 250 epochs is shown in Figure 2.7. In the figure, training performance is measured with training loss and generalization is measured with validation loss. In the beginning of the training, the losses decrease rapidly. After about 25 epochs, overfitting starts as validation set loss starts to increase increases. In order to avoid overfitting and, thus, improve model generalization, regularization methods can be used. Next, we describe regularization methods used for neural networks.

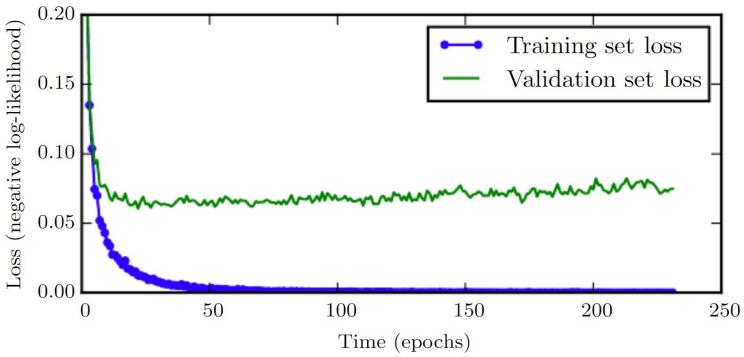



***Figure 2.7.*** *Training process of a network where overfitting occurs [17, p.243].*

### 2.2.3 Regularization Methods

The purpose of regularization is to improve model generalization and avoid overfitting. The bigger the network is in size, the more easily the network is subject to overfitting [6][17, p.225]. In order to train complex models, regularization is required. Next, we describe some common methods which are used for training neural networks in order to improve their generalization and avoid overfitting.

The best way to improve model generalization is to train it with a larger dataset [17, p.236][30]. If more data can not be gathered, it is possible apply transformations to the original dataset and, thus, generate new fake data. Collectively, these transoformation methods are called **data augmentation**. Data augmentation methods adds noise in the

data [17, p.236]. Deep neural networks can be easily overfit to complicated patterns in the dataset. By adding random noise to the dataset for dataset inputs, this restricts the network from learning such complicated patterns [17, p.236]. Data augmentation has been studied to be particularly effective for image classification tasks because of the high input dimensionality of image data [17, p.236]. Because of the high dimensionality, fake images can be generated using a variety of transformation. Some of the operations include, for example, image rotation, scaling and translation [17, pp.236-237].

One of the more recent regularization methods which has been studied to improve generalization is the **dropout** [46]. Normally, during training, the whole network is used in forward propagation to calculate the output of the network. With dropout, connections of the network in the input and hidden layers are dropped out during training using a fixed probability between to sample the dropped connections. Dropout is not applied for output layer. An example of a dropout network is shown in Figure 2.8. In the figure, one neuron in the input layer and two neurons on the hidden layer are dropped out. The result is a thinned version of the original network. During training with SGD, a new thinned network is randomized for every mini-batch with using the layers' fixed dropout percentages [46]. When the error is used to adjust weights of the network, only the weights present in the thinned network are adjusted. The dropout method adds noise to a network's hidden neuron units [46]. This regularizes the training of the network. For model testing, dropout connections are no more used. Dropout is commonly used to train deep neural networks [46][30][44].

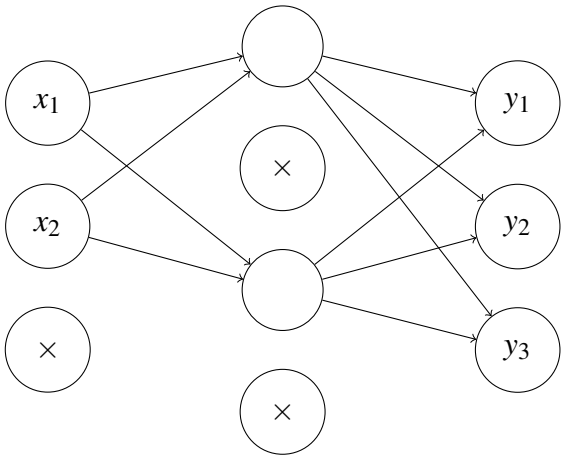



***Figure 2.8.*** *A dropout network.*

One very simple method to avoid overfitting is **early stopping**. Training of neural networks is commonly carried out for a predefined number of epochs. During training, the training and generalization performance are often evaluated after each epoch, for example, by measuring their errors. Usually, the errors decrease rapidly at the start of the training, for example, observe Figure 2.7. At some point during training, the network may start to overfit even though regularization methods are used. At this point, the training process can be stopped so that the generalization performance of the network will not degrade. Early

stopping can be helpful for practical reasons, too, since training deep neural networks can take up to days or even week [44][30]. If network training is well regularized, generalization performance of a model can plateau so that noticeable improvement is no more observed. To save time, the training can be stopped. The early stopping criterion in both of the previously mentioned cases can be, for example, the progress of validation set error. If no validation error decrease is measured for few last consecutive epochs, training can be stopped.

There are many other methods for regularization as well. For object recognition problems, parameter sharing with convolutional neural networks is the most used form of regularization. In parameter sharing, the same weights are used multiple times to detect if a feature is present in the input space rather than having multiple different weights for different positions. Weight sharing decreases network size and, thus, improves its computation efficiency. [17, p.250] In the next section, Section 2.3, we handle convolutional neural networks which are the state of the art method for object recognition [40][30][22].

## 2.3 Convolutional Neural Network

Deep convolutional neural networks have become the state of the art deep learning method for object detection and recognition [31]. Already in 1989, CNNs were successfully applied to recognize hand-written zip code digits [32]. Due to lack of data and computational capacity during that time, CNNs could not be applied for more complex tasks [18]. The modern breakthrough in image classification with deep learning was achieved in ImageNet competition in 2012 [30]. In the competition, *Krizhevsky et al.* improved the benchmark accuracy of the competition by a great margin with their deep CNN implementation [30]. Since then, CNNs have been extensively applied for object detection and recognition tasks [11][25][15][18][22].

For illustration, a CNN model is shown in Figure 2.9. The model classifies color images of a $128 \times 128$ resolution into six categories. A single image contains 49,152 ($128 \times 128 \times 3$) pixels which are the input features of the network. The CNN has one convolutional layer with pooling operation. The input image is filtered with six convolution kernels for feature detection to produce six feature maps. In training, the weights of these kernels are adjusted in order to detect the most meaningful features in the images. Modern deep CNNs may have more than a hundred convolutional layers with hundreds of kernels [22][23][44]. After convolutional layer, the values of the feature maps are flattened to a feature vector which is connected to a FC layer with 30 neurons and then connected to the output layer with six categories. Without FC layers, a CNN architecture would be called a fully convolutional network.

The architecture of a CNN is motivated by its sparse connectivity and parameter sharing. The sparse connectivity is achieved using the convolution kernel which processes only parts of an input at a time. Convolution kernel detects meaningful features, for example,

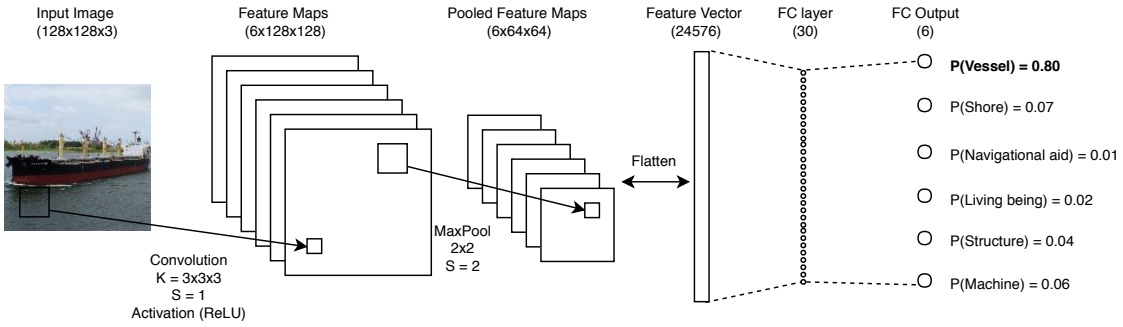



***Figure 2.9.*** *An example of a CNN structure classifying images into 6 different categories.*

edges or other shapes in images. These features are learned during training. The same kernel with the same parameters is used to filter the whole input space. Compared to a traditional fully connected network, use of convolution decreases the number of network parameters, and thus its computational burden. In addition, CNNs often use pooling to further reduce the size of the network.

In order to demonstrate the benefits of convolutional layers, we calculate the number of parameters of the FC layer in the network in Figure 2.9 with and without the convolutional layer. If the input values were connected straight to the FC layer without the convolutional layer, the layer would require 1,474,590 parameters ($128 \times 128 \times 3 * 30 + 30$). With the convolutional layer, the number of parameters is greatly reduced. With six kernels and maxpooling, six feature maps with a resolution of $64 \times 64$ are generated. When these feature maps are connected to the FC layer, only 738,180 parameters are required ($64 \times 64 \times 6 * 30 + 30$). In order to generate the six feature maps, the six convolution kernels require only 60 parameters ($6 * 3 \times 3 \times 3 + 6$). Even a single convolutional layer greatly reduces the number parameters in the network. In order to further reduce the size of the network, more convolutional layers could be used. Next, we describe convolution and pooling operations in more detail.

Convolution operation is a carried out by filtering an input I with a kernel K of the same dimensionality. The product of convolution is a feature map F. [17, p.328] Mathematically, two-dimensional convolution centered at coordinates $(i, j)$ to output a value at the same coordinates into the feature map is defined as

$$F(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{2.12}$$

An example of convolution with a two-dimensional $6 \times 6$ input is shown in Figure 2.10. In color image convolution, the input would be three-dimensional with each one of the dimensions representing one of the color channels of the image. In the figure, two-dimensional input is used for clarity. The convolution is applied with a hand-crafted $3 \times 3$ kernel K and the convolution output is a $4 \times 4$ feature map F. The values of the convolution

kernel are the weight parameters which are adjusted during network training. Typically, the kernel is much smaller than the input [17, p.331]. For example, observe the CNN in Figure 2.9.

In Figure 2.10, the red and blue squares in the input represent a single convolution operations input spaces, and the corresponding squares in the feature map represent the operations outputs. The area of an input which is used to calculate a single value in the feature map, is called the receptive field [17, p.332]. The amount the receptive field moves after each convolution is called stride $S$ [17, p.343]. In the figure, convolution is performed with a stride of one. By increasing the stride, the size of the resulting feature map can be decreased. In this way, the size and computational burden of the network can be reduced. The feature map is formed by applying point-wise multiplication for each kernel position in the input. A single feature map represent features which have been detected with a kernel. Typically, deep CNNs use multiple kernels for feature detection.

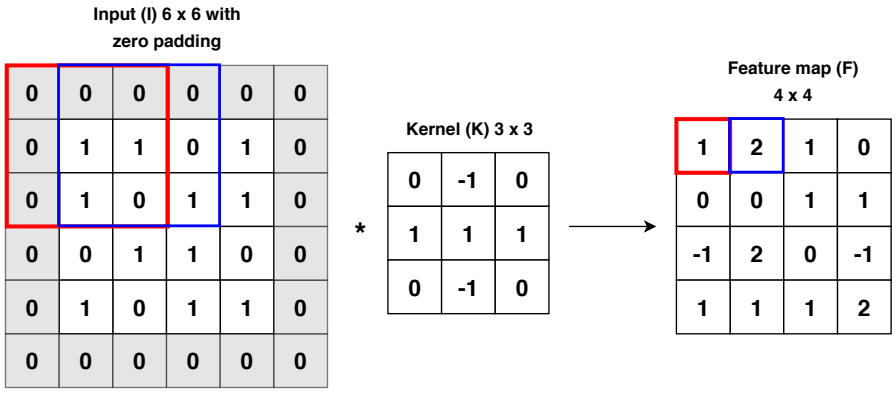



***Figure 2.10.*** *An example of convolution applied for a $6 \times 6$ 2D grid input with zero padding. The red and blue areas represent the input and output of a single convolution operation with the kernel.*

Input padding can also be used to further decrease the computational burden of a CNN. In zero padding, the input is padded with zero values to preserve the spatial ratio of the input for the feature map [17, p.345]. For example, this is performed in Figure 2.10. The original input is a $4 \times 4$ grid. Zero padding is used to increase its dimensions $6 \times 6$ by bordering the input with zero values. Valid padding can be used for network size reduction [17, p.330]. In valid padding, no bordering is applied. This decreases the size of feature map. In the figure, valid convolution would shrink the feature map to a size of $2 \times 2$.

After convolution, feature map values are transformed using an activation function. ReLU is a common choice for deep neural network hidden layer activation [16][30]. In Figure 2.11, ReLU is applied for the previously generated feature map to produce unit outputs.

Some CNN architectures use pooling in order to improve the network's efficiency [44]. Pooling reduces the number of parameters in the network by decreasing the size of feature maps. Pooling operations calculate summary statistics over regions in the input space. Its use is justified when the existence of a feature in the input is more important than its

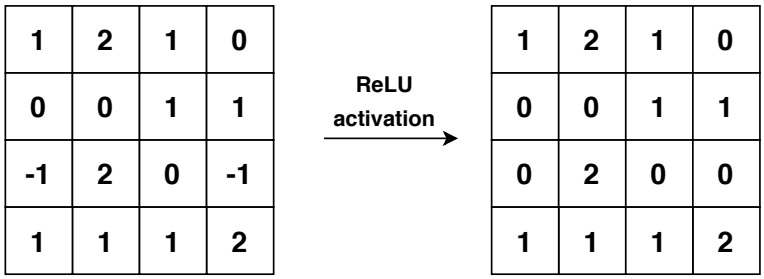



***Figure 2.11.*** *ReLU activation applied for the feature map.*

precise location in input. For example, if we want to detect a face in an image, we are interested in finding face-like features like eyes, nose and mouth rather than their exact position. [17, pp. 335-339] Max pooling is a common pooling operation which is used for CNNs. Max pooling outputs the maximum value within a rectangular area. In Figure2.12, max pooling is applied for size reduction.

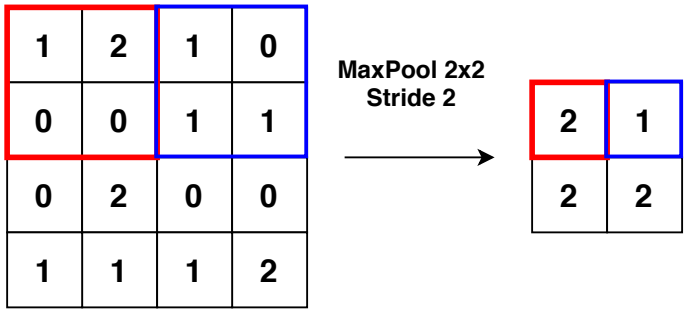



***Figure 2.12.*** *Max pooling is used to downsample the activated feature map.*

In this section, we handled CNNs are their structure. In object detection and recognition tasks, deep CNNs are commonly used [38][47][25][23][51]. Implementing a deep neural network for a task can be time-consuming. Next, we handle practical approaches for training deep neural networks which can be used to facilitate the process.

## 2.4 Practical Methods for Deep Learning

Implementing and training a DNN for a task and to achieve good performance can be time-consuming. In training, there are several hyperparameters which can be adjusted in order to affect the training performance of the network. In addition, much data is required in order to reach good results. Next, we present practical methods which are used for implementing, training and validating DNNs.

### 2.4.1 Hyperparameter Optimization

Parameters of neural networks which can be varied to affect the network's performance i.e. training and generalization are called **hyperparameters**. For example, network architecture, convolution kernel size, learning rate and dropout rate are such hyperparameters.

Learning rate of the optimizer is the most important hyperparameter for a deep neural network. A too large learning rate can result in very poor training error. A too small learning rate can prolong the training and may even result in a poor training error if the optimizer gets stuck in poor local minimum. [17, p.424]

In practice, DNNs can be trained by choosing a set of hyperparameters which values are varied between experiments. After experimentation, training performance of the different models which have been trained with different hyperparameter values can be compared and the model with the best performance is chosen as the final model. The performance metric can be, for example, validation loss or accuracy. The hyperparameters can be tuned manually or with automated processes e.g. grid or random search [17, pp. 427-430]. Next, we present these methods.

In manual tuning, the hyperparameters are tuned manually for every training experiment. This approach can be difficult if the effect of a hyperparameter is not completely understood for training performance. For this reason, automated training processes are useful. In grid search, a list of discrete values is generated for a set of hyperparameters. Each combination of the hyperparameter values is used for training a model. However, grid search may be computationally expensive if there are many hyperparameters which are tuned. As the number of hyperparameters increases by one, the number of combinations increases exponentially. A single training experiment of a deep neural network may take even weeks and some of the tuned hyperparameters may not have an effect for the training performance of the model [44][2]. For these reasons, random search is a better approach. In random search, the hyperparameters values are sampled from a distribution. This way, time is not wasted for tuning hyperparameters which have no effect for model performance. Studies have shown that random search is a better approach for training DNNs compared to grid search when multiple hyperparameters are tuned [2].

### 2.4.2 Transfer Learning

One issue with deep neural networks is the amount of data required for training a model. For supervised deep learning, like image classification, it is estimated, that the training set requires approximately 5,000 labeled examples per category in order to reach an acceptable level of model generalization [17, p.20]. Gathering such a great dataset for a task can be difficult. Transfer learning can be used to decrease the amount of data required for training in order to reach better results.

Transfer learning is a form of supervised pretraining. In transfer learning, the knowledge of a base model is used to improve the performance a new model on with the condition that the base model has been trained for a similar task [50][19][37]. Transfer learning is particularly useful for object recognition tasks with deep CNNs [50][37]. Next, we focus on transfer learning implementation with deep CNNs.

When a deep CNN with multiple layers is trained for image classification, its layers learns

hierarchical features of the images [17, p.6]. The first layers of the network learn more general features such as edges and other abstract shapes. The last layers of the network learn more task specific features. For illustration, a simple CNN used to classify images into three categories is shown in Figure 2.13. The first hidden layer detects abstract edges while the last hidden layer detect more recognizable features of the objects.

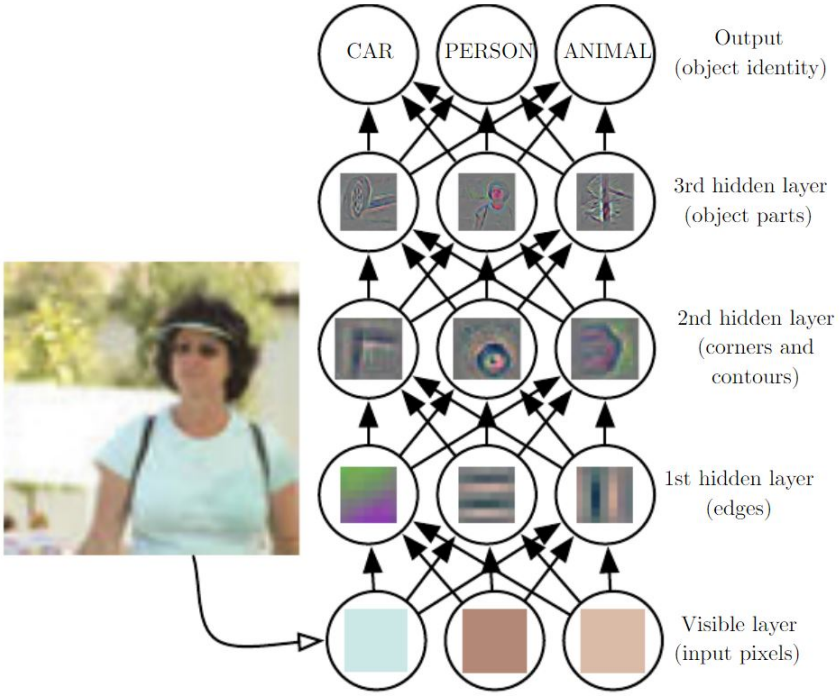



***Figure 2.13.*** *Example of learned features on different layers of a CNN [17, p.6].*

The layers of a trained network can be utilized as the transfer learning base model. The more similar the tasks are, the bigger the impact is for the new model's performance. The base model should be trained with a big enough dataset so that its layers learn general features of the task domain. For image recognition tasks, deep CNN models which have been trained with ImageNet data are good choices as base models. Even though the datasets would be very different, transfer learning has been observed to improve generalization. [50]

There are two strategies for transfer learning implementation, namely feature extraction and fine-tuning. In feature extraction, the layers of the base model are left frozen. The frozen layers are used to extract general features based on the features which were learned in the previous task. During backpropagation, the weights of the frozen layers are not updated. In fine-tuning, the layers of the base model are left unfrozen and thus the weights of the layers are updated. It is possible to use a combination of freezing and unfreezing of the layers. Whether the layers should be left frozen or unfrozen depends on the size of the new dataset and the size of the copied layers. If the size of the new dataset is small and the size of base model layers is great, freezing is preferred to avoid overfitting. For a bigger dataset and a smaller base model, fine-tuning is recommended. [50]

### 2.4.3 Cross Validation

In Subsection 2.1.1, we discussed on the importance of model generalization. Generalization of a machine learning model is measured with a test set which is partitioned from a dataset and not used for training. Estimating model generalization with a simple partitioning can include statistical uncertainty if the dataset is small [17, p.120]. In this case, a model may achieve good performance for the test set by chance even though its real generalization is poor. For more accurate generalization estimation, cross validation methods can be used. Cross validation methods are used to decrease the variance in training results and thus increase the reliability of the results.

One of the most common cross validation method is called k-fold cross validation [17, p.120]. In this method, the dataset is partitioned into k non-overlapping sets/folds, for example, five folds. In training with k-fold, k iterations are performed. In each iteration, a different fold is used to measure model generalization, while the other folds are used for model optimization. The final generalization estimation is the average of k different generalization measurements. When a model is trained with k-fold cross validation, generalization estimation of the model can be improved and the whole dataset information can be utilized for training.

K-fold cross validation can be used for deep learning problems. However, this increases the computational burden of training. Already with a simple train/test split where a model is trained only once, training a deep neural network can take from hours to several days. With K-fold cross validation, the training should be repeated several times. For this reason, deep neural networks are often trained and tested with a simple two part split [9][25][51]. If several models are trained for the same problem, a three part split of training, validation and test set can be used [40]. In addition, in deep learning problems the datasets are often much larger than in traditional machine learning problems. The larger a dataset is, the more it improves the statistical certainty of a simple two part split.

## 2.5 Classifier Evaluation

In this section, methods for evaluating a classifier are presented. For illustration, a two class classification problem of positive and negative examples is presented. A positive example can be classified correctly as positive (True Positive (TP)) or incorrectly as negative (False Negative (FN)). A negative example can be classified correctly as negative (True Negative (TN)) or incorrectly as positive (False Positive (FP)). Given a set of positive examples (P) and negative examples (N), the predictions of a classifier form a confusion matrix. A confusion matrix of the binary classification with positive and negative classes is shown in Table 2.2. [13]

The values of the matrix can be used to calculate different performance metrics for the classifier. One of the most common performance metric for a classifier is its accuracy.

***Table 2.2.*** *A confusion matrix for binary classification.*

| | | True class | |
|---|---|---|---|
| | | Positive | Negative |
| Predicted class | Positive | True Positive | False Positive |
| | Negative | False Negative | True Negative |
| | **Total** | **Positives** | **Negatives** |

Accuracy is simply the number of correctly classified examples divided by the total number of examples. In the two-class classification scenario accuracy is defined as

$$Accuracy = \frac{TP+TN}{TP+FN+TN+FP}. \tag{2.13}$$

If a dataset is imbalanced, evaluating a classifier only by its accuracy can be misleading. With this kind of a dataset, classifying the category with the most examples well and the category with the least of the examples poorly can still lead to good accuracy. However, the confusion matrix can be used to calculate other metrics which are better with an imbalanced dataset. One of them is average class accuracy where the mean of class accuracies is calculated [45].

A classifier can also be evaluated by its receiving operating characteristic (ROC) performance. For this, the classifier's true positive rate (TPR) and false positive rate (FPR) are required [13]. In machine learning, TPR is also called hit rate or recall. It is the proportion of correctly predicted positives to all positive predictions. It has a value between 0 and 1. In the binary classification, true positive rate is defined as

$$TPR = \frac{TP}{TP+FN} = \frac{TP}{P}. \tag{2.14}$$

FPR is the opposite of TPR. FPR is the proportion of incorrectly classified negatives to all negative predictions. It has a value between 0 and 1. In the binary classification, false positive rate is defined as

$$FPR = \frac{FP}{FP+TN} = \frac{FP}{N}. \tag{2.15}$$

Performance of four different classifiers is plotted to ROC space shown in Figure 2.14. Several points in the ROC space play important part for evaluating a classifier. The point (1,0) represents a perfect classifier. In it, all positive examples are correctly classified as True Positive (TPR=1) and none of the negative examples are incorrectly classified as False Positive (FPR=0). In point (1,1), the classifier classifies all positive examples correctly (TPR=1), however, it also classifies all negative examples as positive (FPR=1). In point (0,0), the classifier predicts all examples as negative (TPR=0, FPR=0). [13]

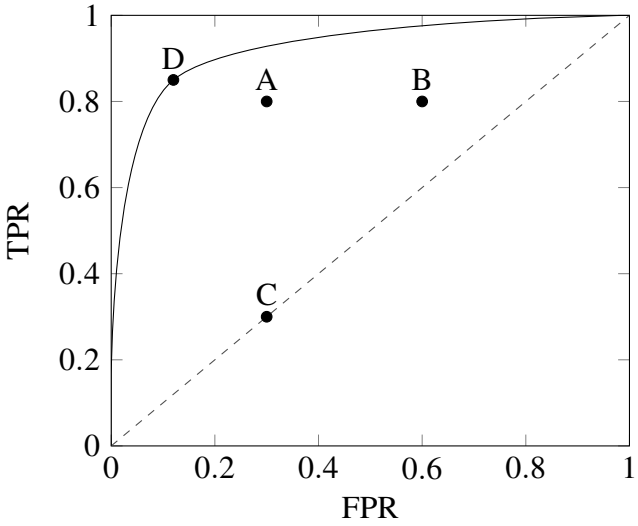



***Figure 2.14.*** *ROC space and performance of three discrete classifier and one probabilistic classifier.*

Discrete classifiers, such as decision trees, output discrete categories without initial values when predicting a category. The performance of discrete classifiers can be plotted as a single dot in the ROC space. In Figure 2.14, three discrete classifiers A, B and C are plotted in ROC space. Classifier A predicts positive examples correct 80% but has a false positive rate of 30%. Classifier B has the same TPR as A but is worse in predicting negative examples with a FPR of 60%. In the figure, the dashed black line represents performance of a random classifier. Classifier C is as good as a random classifier. It classifies only 30% of positives correct but false positive rate is only 30%. Any reasonable classifier should be located above the dashed black line. Anything below it has a worse performance than a random classifier. From the three classifiers, classifier A performs the best. The closer a classifier is to the upper left corner in the ROC space, the better its performance is.

Some classifiers, such as neural networks, output initial probabilities for output categories which are used to make the final classification decision. Performance of these classifiers can be evaluated by plotting their ROC curve by comparing the output values with classification thresholds. If an output value is higher than the threshold, it is classified as positive. If an output value is lower than the threshold, it is classified as negative. With every classification threshold, a point in ROC space can be plotted. By sliding the classification threshold value from one end to the other, the ROC curve of the classifier is plotted. The ROC curve allows to evaluate the performance of a classifier with different thresholds. In Figure 2.14, the black line represents a ROC curve of a classifier. With the ROC curve, the performance of the classifier can be tuned to exhibit wanted classification behaviour with respect to the TPR and FPR rates. At some threshold value of the classifier, it achieves a TPR of 85% and FPR of 12%. The classifier with the threshold value is plotted as classifier D at the ROC curve of the classifier.

A major benefit of plotting a ROC curve is its invariance for dataset class imbalance. When

a classifier is evaluated with accuracy, the accuracy represents the score for the particular distribution in the dataset. However, ROC curve is unaffected with dataset class imbalance for which reason it is a good performance metric. [13]

An important metric which is derived from ROC curve is the area under curve (AUC). AUC is, as the name implies, area under the ROC curve and has value between 0 and 1. Any reasonable classifier should have AUC value over 0.5 because this is the threshold a random classifier is able to achieve. The value of AUC is equivalent to the probability of the classifier to predict a random positive example with greater confidence than a random negative example. With the AUC, performance of different classifiers can be easily compared. [13]

ROC and AUC concept can be expanded for problems with more than two classes. In a multiclass scenario, ROC curve and AUC can be calculated for each class separately. This is done by plotting ROC curve of each class as binary classification where the observed class is assigned as positive class and the rest of the classes are assigned as negative class for that time. The average classification performance of a multiclass classification can be evaluated by its macro-average ROC curve which is preferred if the classes are to be treated equally [45]. A point in the marco-average ROC curve is calculated by summing the TPR and the FPR rates of each class with a given threshold and by dividing them with the number of classes [45]. With the marco-average ROC curve, average AUC score of the classifier can be calculated.

## 2.6 Related Work

In the thesis experiments, we implement deep CNNs for image classification with RRMI dataset. The focus of the experiments is to consider the use of deep CNNs for real-time object recognition in maritime environment. In this chapter, studies related to the thesis research are reviewed. First, we present results which have been achieved in ImageNet competition for object recognition. Then, other studies which are more related to maritime environment are presented.

ImageNet is shortening for ImageNet Large Scale Visual Recognition Challenge (ILSVRC) which is an annually held competition in the field of object detection and recognition since 2010. The competition consists of tasks in image classification and object detection. The image classification competition consists of a dataset of 1.2 million images in 1,000 categories. The competition has acted as a benchmark for the latest advances in the field of object recognition. [40] In order to give a perspective for deep learning development in image classification, we review some of the models which have been able perform particularly well in the competition. In the competition, performance is measured in top-1 and top-5 errors. Top-1 error is proportion of incorrectly classified examples. Top-5 error is measured by checking if the correct prediction is one five highest predictions of the classifier.

Year 2012 marks a breakthrough for deep learning and image classification. In ILSVRC-2012, *Krizhevsky et al.* made a breakthrough in the competition with their deep CNN model, AlexNet, by increasing the benchmark accuracy by great margin. The deep CNN architecture consisted of convolutional and fully connected layers with 60 million parameters. For downsampling, the architecture used strided convolution and maxpooling. ReLU activation was used throughout the hidden layers of the network. Data augmentation and dropout were used for regularization. For training, parameters of the network were initialized with random weights. For optimization, SGD was used by manually tuning the learning rate during training. The model was trained for six days with NVIDIA GTX 580 3GB GPUs. The model achieved a top-1 error of 37.5% and top-5 error of 17.0%. [30]

In ILSVRC-2014, *Simonyan et al.* were one of the top teams in the competition. In their research, they trained multiple CNN models with up to 19 weight layers. The architectures consisted of convolutional layers with fully connected layers with up to 144 million parameters. Convolutions were carried out with kernels of $3 \times 3$ and $1 \times 1$ and maxpooling was used for downsampling. ReLU was used for hidden layer activation. The models were trained with a combination of random weight initialization and pretraining. Training of the models was performed with SGD with weight decay. The learning rate was adapted during training when validation accuracy stopped improving. Data augmentation and dropout regularization were used to improve generalization. Training was performed with 4 NVIDIA Titan Black GPUs processing in parallel. A single model trained for two to three weeks depending on the depth of the architecture. The best top-1 and top-5 errors of 23.7% and 6.8%, respectively, were achieved using an ensemble of two of the deepest models. One of them was **VGG16** which is used later in the thesis for training a classifier. [44]

Year later in ILSVRC-2015, the benchmark was improved by *He et al.* with deep residual CNNs (ResNets). In their research, they trained multiple CNN architectures varying from 18 weight layers to as deep as 152 weight layers with ReLU activation. Unlike with other CNN architectures, downsampling in ResNets is mostly performed with strided convolution. Also, the network was a fully convolutional network. Convolutions are performed with kernel sizes of $3 \times 3$ and $1 \times 1$. The network consists of approximately 25 million weights (reported with Keras model summary) [8]. The layers of the deeper networks were initialized using the pretrained layers of smaller networks in order to improve their generalization. The architecture also expanded the basic CNN concept by adding shortcut connections, residual connections, to the network. They demonstrated that these shortcut connections improve the training performance for deeper models compared with plain networks without the shortcuts. For training, SGD was used with error adaptive learning rate. Weight decay and data augmentation were used for regularization. A top-5 error of 3.56% was achieved an ensemble of six ResNet models of different depth. One of the ensemble models, **ResNet50**, achieved a top-1 error of 20.74% and top-5 error of 5.25%. The ResNet50 architecture is later used in the thesis for training a model. [22]

*Howard et al.* have developed a CNN architecture called **MobileNet**. The model has not

been an entry in ILSVRC competition but has proven useful otherwise. Much of the latest research with DNNs is motivated only by increasing the accuracy disregarding the model size and computational requirement. MobileNet research has been motivated by regarding these factors, which are important real-time time classification applications. MobileNet CNN architecture size is configurable. The biggest configuration of MobileNet consist of approximately 4.2 million weights in 28 weight layers. The architecture consists of normal and depthwise convolutions, which greatly reduces the size and computational requirement of the model without remarkable change in accuracy. Downsampling is mostly performed with strided convolution operations with a kernel size of $3 \times 3$. The MobileNet architecture was trained to classify ImageNet data to evaluate its performance. Contrary to bigger models such as VGG16 and ResNet50, MobileNet uses much less regularization methods when trained. This was justified by the small size of the model, which restricts it from overfitting. The biggest configuration of MobileNet (1.0 MobileNet-224) achieved an accuracy of 70.6%. This configuration of the MobileNet architecture is used later in the thesis for training a classifier. [23]

*Zhang et al.* have conducted research for marine object recognition with a focus on autonomous vessels. Their research is motivated by automating the seaborne shipping industry which is estimated to be worth 375 billion dollars. The operating costs of the industry are estimated to cover 44% of all the expenses. In their research, they built a system capturing color and infrared images of vessels. The system was used to capture images of vessels during day and night. The images were labeled into 6 different categories: merchant, sailing, passenger, medium, tug and small. The final VAIS dataset consists of 2,865 images with varying resolution, 1,623 in color and 1,242 infrared with 1,088 corresponding pairs. In their experiments, they were able to achieve an accuracy of 81.9% for color image classification using transfer learning with VGG16 architecture. For IR image classification, they reached an accuracy of 59.9%. [51]

*Dao et al.* have researched vessel classification with a focus on safety and security of the maritime sector. In their research, they gathered images of vessels from a vessel spotting website. The final dataset, E2S2-Vessel, consisted of 130,000 vessel images in 35 categories. Configurations AlexNet CNN architecture with 58 and 26 million parameters were used for training without pretraining. Training was carried out using SGD optimizer with adaptive learning rate with NVIDIA GeForce GTX 560 Ti GPU. The best model was able to classify images with an accuracy of 80.91%. Training of the models lasted for over 24 hours. They also evaluated the computational performance of the models. Their deep CNN models were able to classify images in close to 4 milliseconds. [9]

# 3. RESEARCH DATA AND METHODS

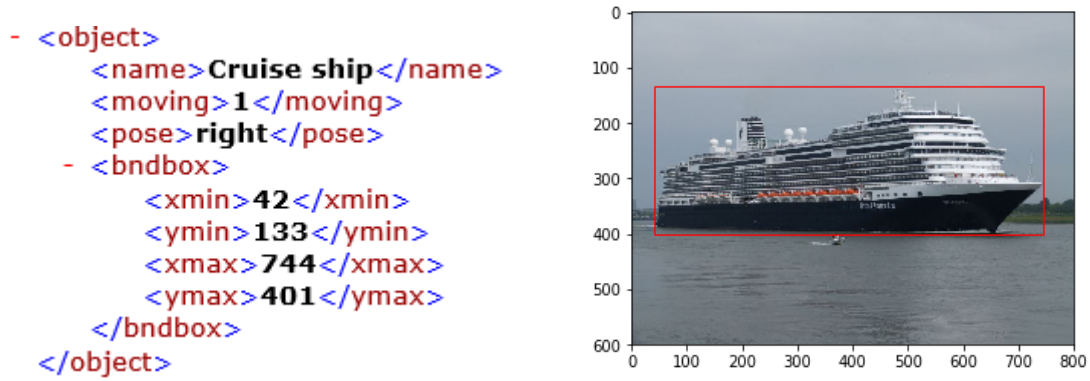
In the experiments, we train deep neural networks for image classification. The focus of the experiments is to study object recognition using deep neural networks and evaluate its potential for a real-time maritime application. In this chapter, the research data and the training implementation methods which are used in the experiments are described. First, we present the research data. Then we provide details of model implementation and present the methods used for training and evaluating the models.

For research implementation, Python programming language [39] was used for programming. SciPy [28] and NumPy [48] packages were used for data preprocessing and formatting. The results were plotted using Matplotlib [24]. The neural networks were modeled and trained with Keras [8] using Tensorflow [1] backend. Keras is a framework which enables an easy and fast implementation, and experimentation for neural network models. Tensorflow is an open source library which enables a highly efficient computation with distributed systems. It can be used to train neural networks with Keras using a GPU which greatly reduces the time required for training them. The experimentation and training of the models was performed using a local PC with Windows 10 64-bit operating system. The hardware components of the PC included: NVIDIA GeForce GTX 1080 8GB GPU, Intel i7-3770 CPU and 16 GB of RAM.

## 3.1 Research Data

In this section, the research data set and its preprocessing is described. The research data for this project was provided by Rolls-Royce. Before preprocessing, the dataset consisted of 8,000 images. Each image had an XML-annotation of objects and their location in the image. One image contained one or multiple objects. An example of an XML-annotation of Cruise Ship bounding box information and its location in the image is shown in Figure 3.1. The resolutions of these objects in the images varied from the smallest resolution of $15 \times 13$ pixels to the highest resolution of $732 \times 490$ pixels.

The object classes have a hierarchical form. The class hierarchy was formed by a separate XML-file which was provided by Rolls-Royce. For example, Cruise ship class in Figure 3.1 is a subclass of Passenger vessel class which is a subclass of Vessel class. With XML-file, the classes of the objects were mapped to the top hierarchy level. After mapping, the dataset was cleaned. It was found, that some of the images were black and white. These images were removed from the dataset. This was done to avoid problems in inputting information to the neural networks as the dimensions of all of the inputs have to match. Also, classes with less than 100 images were removed from the dataset. The final composition of the




(a) XML-annotation (b) Object bounding box

***Figure 3.1.*** *Example of an object XML-annotation (a) and bounding box location of the object in its corresponding image (b).*

dataset used in the experiments and its classes is shown in Table 3.1. This dataset is called Rolls-Royce Maritime Image (RRMI) dataset.

***Table 3.1.*** *RRMI dataset classes and number of images.*

| **Class** | **Number of images** |
|---|---|
| Living being | 1,423 |
| Machine | 1,013 |
| Navigational aid | 814 |
| Shore | 14,023 |
| Structure | 1,857 |
| Vessel | 11,845 |
| **Total** | 30,975 |

For image extraction and resizing, a similar approach to [11] was used in the experiments. For training the models, the images of objects were extracted from the images as separate images and resized to the resolution which matched the input dimensions of the used network. The initial bounding box area was made square shaped for extraction. This was done by increasing the shorter side of the bounding box equally to both axis directions in order to match to the longer side. In this way, the original aspect ratio of an image is preserved when the image is resized. Without increasing the dimensions, a worse performance is achieved when training a CNN classifier [11]. In case the new bounding box area extended over the image area, image size was increased recreationally. This was performed using OpenCV [4] *copyMakeBorder* function's *border reflect* method which mirrors the borders of an image to increase its resolution. For resizing, nearest neighbour interpolation was used. Three sample images of each class in RRMI dataset resized to a matching resolution are shown in Figure 3.2. Because the initial resolution of the images varied, some of the images are sharper while some of the images are more fine-grained.

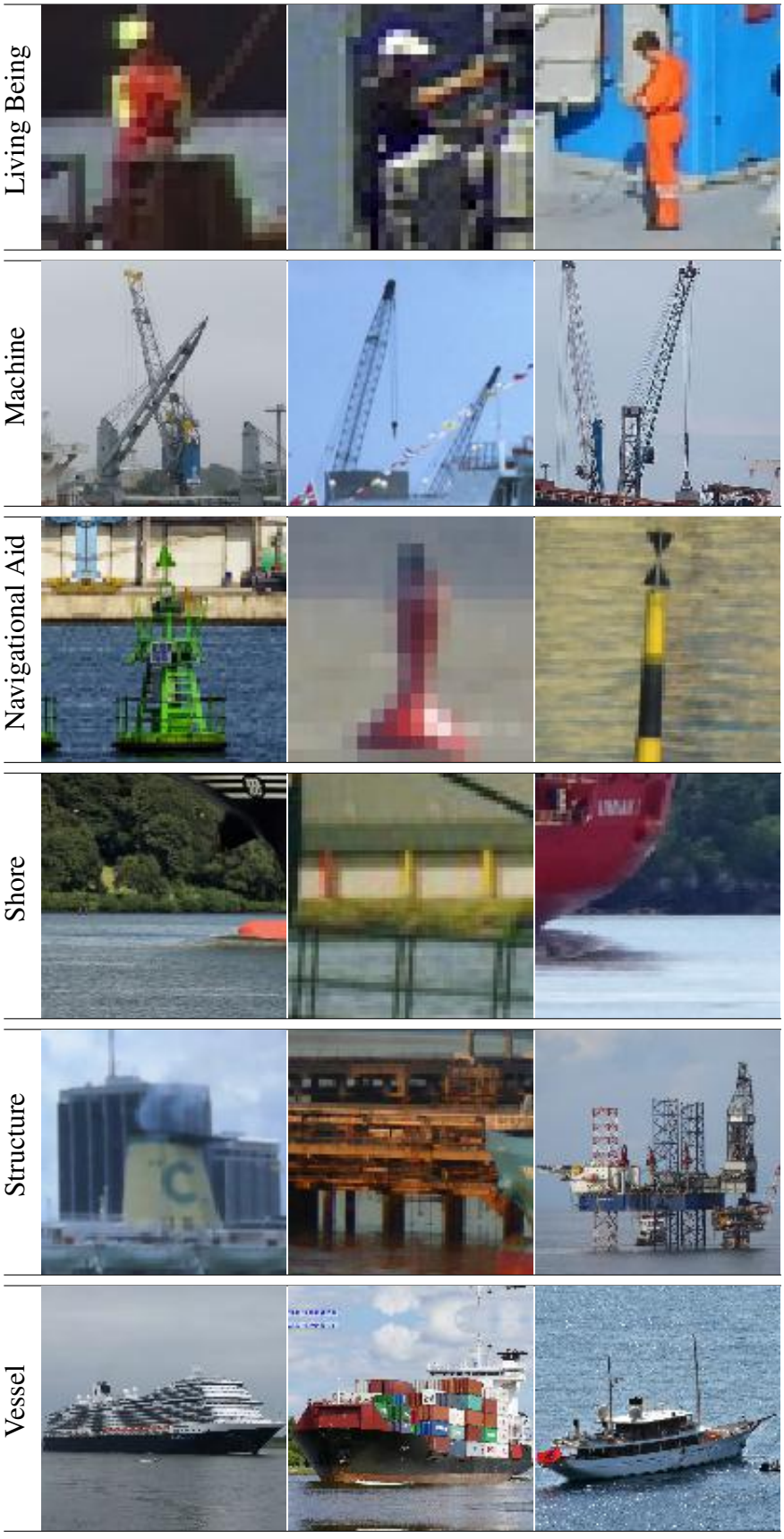



***Figure 3.2.*** *Three sample images of each class in RRMI dataset.*

## 3.2 Model Implementation

For model implementation, two distinct approaches are used. First, we train a deep CNN using a random hyperparameter search approach. Then popular deep CNN architectures are trained using transfer learning. In this section, we provide an introduction to the used methods. In the following subsections, we describe the methods used for training and evaluating the models.

Training a DNN for a task can be time-consuming as discussed in Section 2.4.1. There are many hyperparameters which can be tuned to affect the training performance. In the first experiment, Small-CNN network architecture and its training parameters are randomized using an automated random hyperparameter search approach. Before training, the weights of the network are initialized randomly. The hyperparameters are randomized for 50 iterations. After 50 iterations, the model with the highest validation accuracy is chosen to be further analyzed. The approach is similar to work in [25] where random hyperparameter search was performed for generating and training the network.

Network architectures which have performed well with ImageNet are often trained for other image classification problems as well [51][9][19]. The Keras framework allows to use ready-made versions of some well performing architectures. In the following experiments, VGG16 [44], ResNet50 [22] and MobileNet [23] architectures are trained for classification. These architectures were described in Section 2.6. In addition, these architectures are initialized with ImageNet pretrained weights in order to train them with transfer learning. The Keras framework's implementation of the architectures and the pretrained weights are used. For training, the base models are left unfrozen and thus fine-tuned in training. Also, other minor modifications are done to the architectures before training. These modifications are explained in more detail in the experimental chapter.

### 3.2.1 Training Methods

For training, the RRMI dataset is partitioned into training and validation sets in a stratified manner with a 80%/20% split. Training set consists of 24,870 images and the validation set 6,218 images. Training set is used for model optimization and validation set is used to estimate model generalization. The split procedure followed the example of many DNN implementations, e.g. in [25][9][51]. For training, the images are resized to resolutions matching the architectures.

During training, loss and accuracy of both training and validation set are measured after each epoch. Loss is measured using cross entropy (Equation 2.9). For this, the dataset target outputs were binarized as explained in Section 2.2.2. For classification, softmax activation is used to output class probabilities. The class with the highest probability is chosen without a specific classification threshold.

The RRMI dataset is imbalanced in class distribution. We want to train the classifiers with equal class importance. For this, a cost-sensitive method can be used [21]. With

this method, the weights of the network are updated by the inverse proportion of the class examples in dataset:

$$u_k = \frac{N}{Kn_k}, \tag{3.1}$$

where $u_k$ is the weight update factor for output class $k$, $N$ is the total number of examples in the dataset, $K$ number of classes and $n_k$ number of examples for class $k$. This way, the weight updates of minority classes have a greater effect than those of the majority class examples. In training, the cost-sensitive method was implemented using *sklearn*'s class weight package's *balanced* option.

For network weight optimization, SGD with a mini-batch size of 32 with the cost-sensitive approach is used. A power of 2 mini-batch size is used for a improved training performance with the GPU [17, p.276]. Training data is augmented by mirroring the training set images in order to improve generalization. The networks are trained for a maximum of 50 epochs. Early stopping is used to terminate training process if validation loss decrease was not observed for five consecutive epochs in order to prevent overfitting. The models are trained by various learning rates. In the random hyperparameter search experiment, the learning rate is randomized. With VGG16, ResNet50 and MobileNet, the learning rate is tuned with grid search.

### 3.2.2 Evaluation Methods

In the experiments, the classification and computational performance of the models is evaluated. From each training approach, the best model is chosen. The best model is chosen according to the validation accuracy after training. In total, four different models and their performance are compared. These include Small-CNN, which architecture is randomized, VGG16, ResNet50 and MobileNet architecture models. Next, we describe the methods and metrics used for model evaluation.

After training, generalization performances of the models are compared with their validation loss and accuracy. Accuracy is a commonly used metric for image classification [44][51][9][25]. In addition, we compare the validation losses of the models. With softmax classification and cross entropy loss error function, training and generalization performance of a model can also be evaluated by comparing the loss values of different models.

We evaluate the classification performance of each class in the RRMI dataset. For this, a confusion matrix of the validation set predictions is generated and the values are normalized. These normalized values are used to observe the classification performance of each category in the dataset with a given model. Using these normalized classification accuracies of each class, average class accuracy of a model is calculated by taking the mean the class accuracies. The average class accuracy is calculated in order to evaluate the cost-sensitive

training implementation. In addition, it is used to evaluate classification performance with other than pure accuracy metric. This is because classification with an imbalanced dataset can be misleading [5][21].

Classification performance of the model with the best average class accuracy is assessed and evaluated with ROC curves. In the experiments, no classification threshold is used. The output class is chosen according to the maximum value of the output vector. If a threshold was used, the classifier could be tuned for ideal performance using its ROC curves. For example, the threshold could be selected so that the classifier would be very confident of its Vessel classifications by having a high hit rate and low false positive rate. In the experiments, ROC curve of each class was plotted as binary classification scenario. The AUC scores were calculated in order to compare the classification performance of each class in the dataset with various thresholds.

For a real-time object recognition application, computational performance is important [9]. In the experiments, computational performance of the models is evaluated by measuring the time required for image classification. For this, 100 images are classified with the models and the time is measured. Time is measured using Python's *time* module. The measurements are performed using both the CPU and GPU.

# 4. EXPERIMENTAL RESULTS

In this chapter, the results of the thesis experiments are presented and evaluated. The research data and methods used for experimentation were described in the previous chapter. First, we present the training results of the single experiments. In the end of the chapter, we evaluate and compare the classification and computational performance of the models.

## 4.1 Training Results

In this section, the results of the experiments are provided. First, the results of random hyperparameter search experiment which results in Small-CNN model are presented. This is followed with results of VGG16, ResNet50 and MobileNet models which were trained with transfer learning using ImageNet pretrained weights.

### 4.1.1 Small-CNN

CNN architectures were generated and trained using a random hyperparameter search approach. The hyperparameters included input image resolution, network architecture parameters and learning rate. The randomized hyperparameters, and their range are shown in Table 4.1. After 50 iterations, the CNN with the best validation accuracy was chosen. This model is called Small-CNN. The values of the hyperparameters values of Small-CNN are shown in the right-hand side in Table 4.1. Interestingly, all of the hyperparameters which affected the size of the model except kernel size have the maximum values. Kernel value is one step from the maximum value. The training has favored a bigger model. It is possible that all parameters would have been maximized if more than 50 iterations were performed.

***Table 4.1.*** *Hyperparameters, their range of values and the selected values with random search approach. Learning rate is sampled from an uniform distribution.*

| **Hyperparameter** | **Value Range** | **Selected Value** |
|---|---|---|
| *Convolutional layers* | $\{1,2,3,4\}$ | 4 |
| *Fully connected layers* | $\{1,2\}$ | 2 |
| *Kernel size* | $\{3,5,7,9,11\}$ | 9 |
| *Feature maps* | $\{16,32,48,64\}$ | 64 |
| *Learning rate* | $10^{-6} - 10^{-3}$ | 0.000799 |
| *Input size* | $\{32,64,96,128\}$ | 128 |

A detailed description of the architecture of Small-CNN can be observed in Table 4.2. For input, images were resized to a resolution of $128 \times 128$. The input layer was connected

to 4 convolutional layers. Each of the convolutional layers generated 64 feature maps with $9 \times 9$ convolution kernels. The convolutions were carried out with zero padding and stride of one. Maxpooling was used for downsampling. The feature maps of the last convolutional layer were flattened to a feature vector and connected to two FC layers before output layer. For regularization, dropout connections with a 25% dropout rate were used after FC layers. The weights of the network were initialized randomly using the default initialization settings of Keras. ReLU activation was used throughout the hidden layers of the network. The network consisted of a total of 1,552,838 parameters.

***Table 4.2.*** *The architecture of the Small-CNN. K stands for kernel size and FM for number of feature maps.*

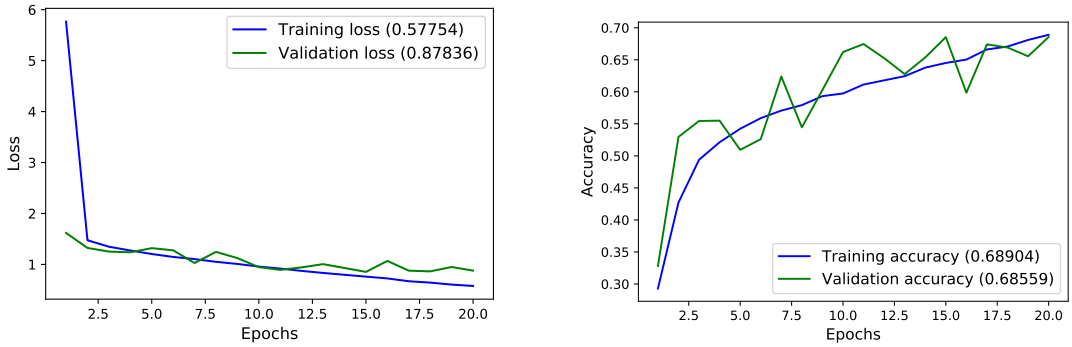
| **Layer** | **Activation** | **FM** | **K** | **Input size** | **Output size** | **Parameters** |
|---|---|---|---|---|---|---|
| Input | | | | $[128 \times 128 \times 3]$ | $[128 \times 128 \times 3]$ | 0 |
| Conv1 | ReLU | 64 | $[9 \times 9 \times 3]$ | $[128 \times 128 \times 3]$ | $[128 \times 128 \times 64]$ | 15,616 |
| Maxpool1 | | | | $[128 \times 128 \times 64]$ | $[64 \times 64 \times 64]$ | 0 |
| Conv2 | ReLU | 64 | $[9 \times 9 \times 64]$ | $[64 \times 64 \times 64]$ | $[64 \times 64 \times 64]$ | 331,840 |
| Maxpool2 | | | | $[64 \times 64 \times 64]$ | $[32 \times 32 \times 64]$ | 0 |
| Conv3 | ReLU | 64 | $[9 \times 9 \times 64]$ | $[32 \times 32 \times 64]$ | $[32 \times 32 \times 64]$ | 331,840 |
| Maxpool3 | | | | $[32 \times 32 \times 64]$ | $[16 \times 16 \times 64]$ | 0 |
| Conv4 | ReLU | 64 | $[9 \times 9 \times 64]$ | $[16 \times 16 \times 64]$ | $[16 \times 16 \times 64]$ | 331,840 |
| Maxpool4 | | | | $[16 \times 16 \times 64]$ | $[8 \times 8 \times 64]$ | 0 |
| Flatten | | | | $[8 \times 8 \times 64]$ | 4096 | 0 |
| FC5 | ReLU | | | 4096 | 128 | 524,416 |
| Dropout | | | | | | 0 |
| FC6 | ReLU | | | 128 | 128 | 16,512 |
| Dropout | | | | | | 0 |
| Output | SoftMax | | | 128 | 6 | 774 |
| | | | | | | **1,552,838** |

The training performance of the Small-CNN is plotted in Figure 4.1. The training was terminated after 20 epochs with the early stopping policy. After final epoch, the model reached a training accuracy of 68.9% and a validation accuracy of 68.6%.

The normalized confusion matrix of the validation set predictions with the Small-CNN is shown in Figure 4.2. Navigational aid had the highest classification accuracy of 82%. Living being, Machine, Shore and Vessel were predicted correctly with rates between 70% to 71%. The poorest accuracy of 34% was achieved with Structure images. Observing the figure, Structure images were very often classified as Shore. Also, Shore images were predicted incorrectly as Structure in 14% of the cases.

### 4.1.2 VGG16

The pretrained VGG16 architecture was modified and trained to classify the RRMI images with transfer learning. All of the pretrained FC layers and output layer with ImageNet outputs were removed. These were replaced with a FC layer with 256 neurons with ReLU activation which was connected to the output layer with softmax classification. Also, dropout connections with a dropout rate of 25% were added after the FC layer for




(a) Loss (b) Accuracy

***Figure 4.1.*** *Training performance of Small-CNN.*

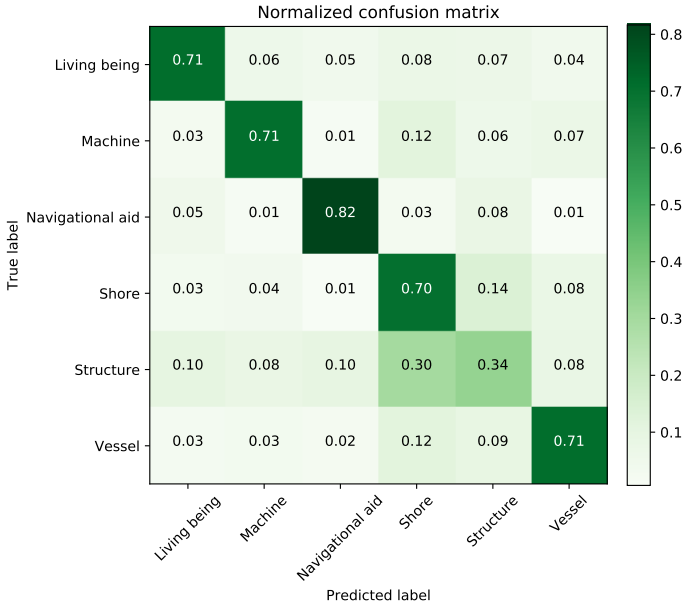
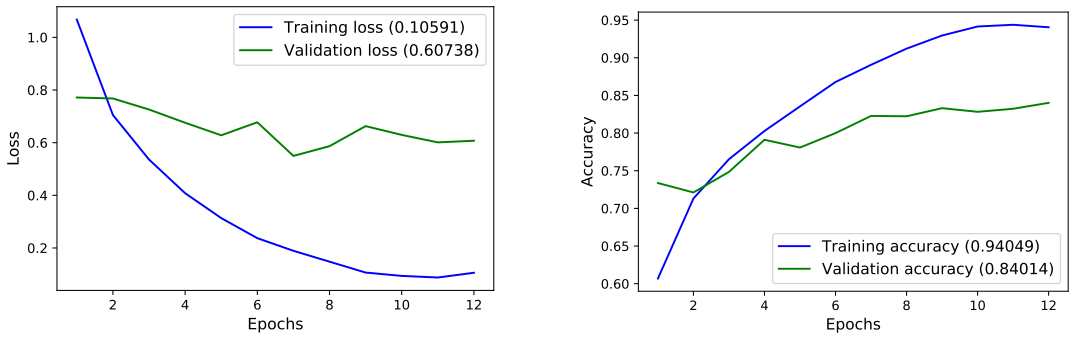



***Figure 4.2.*** *Small-CNN's normalized confusion matrix.*

regularization. The modified VGG16 model consisted of 21 million parameters. Seven models with learning rates between $10^{-7} - 10^{-1}$ with exponential steps of 1 were trained. Models with the greatest learning rates were observed to get stuck in training while models with lower learning rates were observed to converge very slowly. The best validation accuracy of was achieved with a learning rate of $10^{-3}$. The training process of this model is shown in Figure 4.3. The model was trained for 12 epochs before early stopping was used. The model reached a training accuracy of 94.0% and validation accuracy of 84.0%.

The normalized confusion matrix of the validation set predictions with the VGG16 model is shown in Figure 4.4. The model classified all classes except Structure with an accuracy of $76\% - 88\%$. The same misclassification problem of Structure as Shore was present with VGG16 model. VGG16 model misclassified Structure as Shore with a rate of 38%.




(a) Loss (b) Accuracy

***Figure 4.3.*** *Training performance of VGG16.*

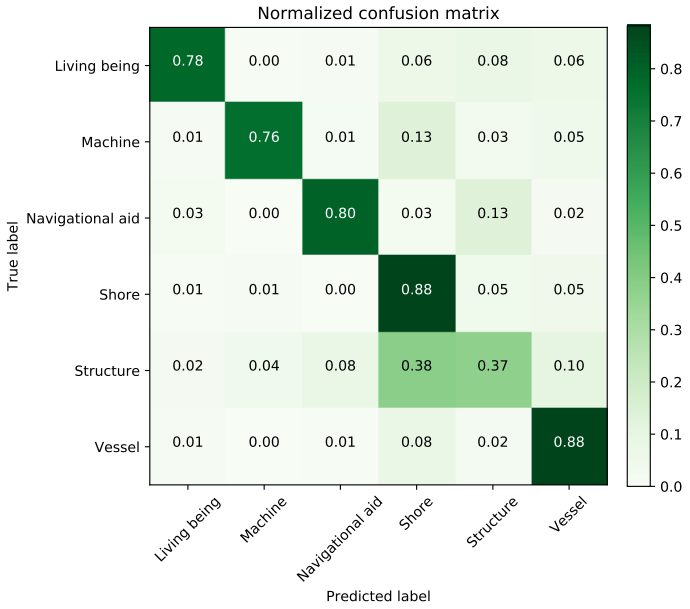


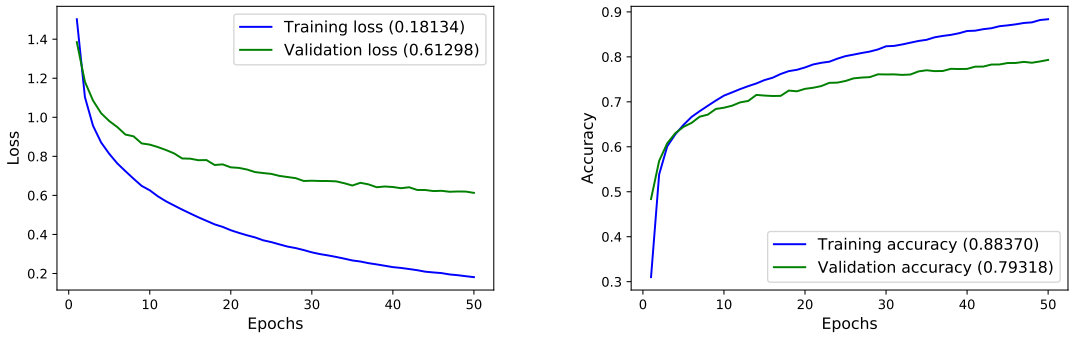

***Figure 4.4.*** *VGG16's normalized confusion matrix.*

### 4.1.3 ResNet50

The pretrained ResNet50 architecture was modified for training implementation. The output layer of network was removed. A FC layer with 512 neurons with ReLU activation. The FC layer was connected to the output layer with softmax activation. Dropout connections with a rate of 25% were added after FC layer for improved training regularization. The modified model consisted of 24.6 million parameters. Two models were trained with learning rates of $10^{-3}$ and $10^{-4}$. A better validation accuracy was achieved with the latter. Training process of the better model is shown in Figure 4.5. Compared to the two previous models, the model trained for the maximum 50 epochs and no early stopping criterion was met. It achieved a training accuracy of 88.4% and validation accuracy of 79.3%.

The normalized confusion matrix of the validation set predictions with the ResNet50 model is shown in Figure 4.6. The model achieved an accuracy of 78%−87% for all the classes

except Structure for which an accuracy of 50% was achieved. The same misclassification problem of Structure being classified as Shore was present with ResNet50 model.




(a) Loss (b) Accuracy

***Figure 4.5.*** *Training performance of ResNet50.*

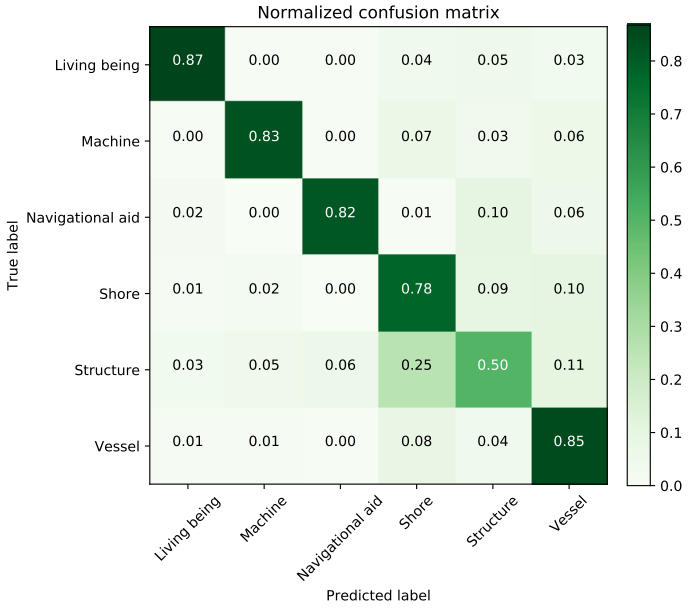
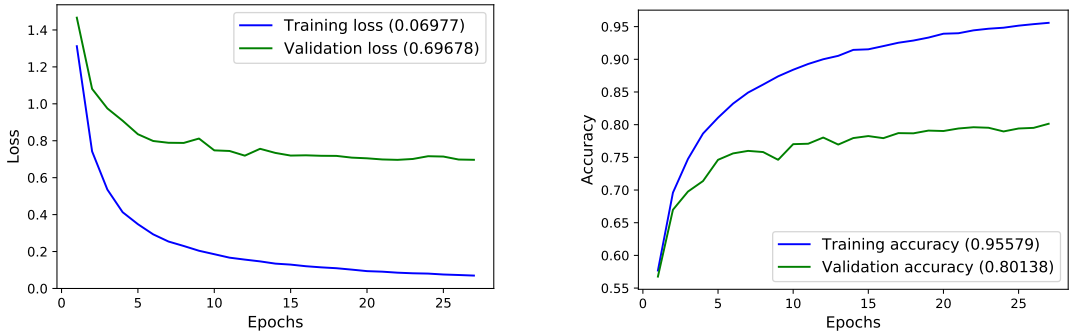



***Figure 4.6.*** *ResNet50's normalized confusion matrix.*

### 4.1.4 MobileNet

The pretrained MobileNet architecture was used for trained. The only modification for the network was the replacement of ImageNet output layer with an output layer for RRMI classification. The modified MobileNet model had 3.5 million parameters. Learning rates of $10^{-3}$, $10^{-4}$ and $10^{-5}$ were used for training MobileNet model. The best validation accuracy was achieved with the learning model of $10^{-4}$. The training process of the model is shown in Figure 4.7. Training was performed for 27 epochs before early stopping criterion was met. A training accuracy of 95.6% and validation accuracy of 80.1% were achieved.

The normalized confusion matrix of the validation set predictions with the MobileNet model is shown in Figure 4.8. The classification performance is similar to previous experiments. While Living being, Machine, Navigational aid, Shore and Vessel had an accuracy between 73%−84%, Structure had only an accuracy of 30% and was more often misclassified as Shore.




(a) Loss (b) Accuracy

***Figure 4.7.*** *Training performance of MobileNet.*

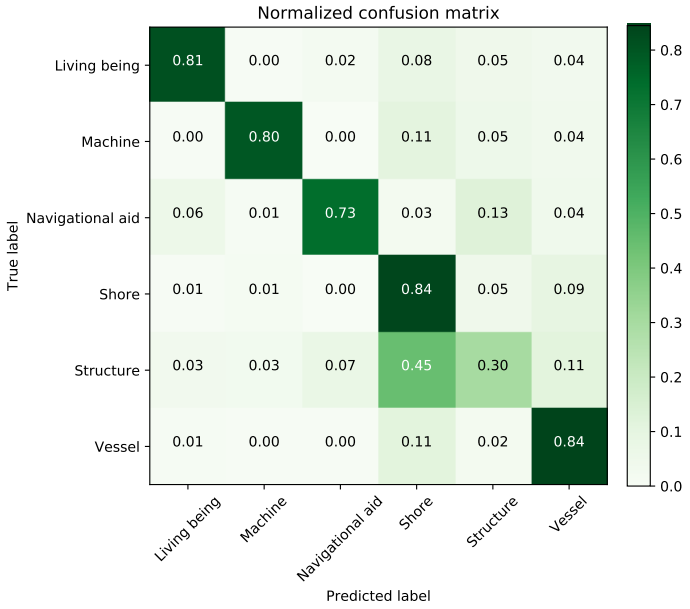



***Figure 4.8.*** *MobileNet's normalized confusion matrix.*

## 4.2 Model Evaluation

In this section the training, classification and computational peformances of the models are comparatively evaluated and analyzed. First, we compare the training results and implementation of the models. Second, we evaluate the classification performance of the models and calculate the average class accuracy of the models. Finally, we analyze the misclassification problem of Structure which was observed during the experiments. The classification performance of the model with the highest average class accuracy is further

evaluated by plotting its ROC curves. In the end of the section, computational performance of the models is evaluated.

### 4.2.1 Training Performance

Training and validation results of the models were gathered to Table 4.3. Overall, the models which were trained with transfer learning performed well. The MobileNet model was the best model in fitting the training data. It achieved the smallest training loss of 0.06977 with a training accuracy of 95.6%. The VGG16 had the best validation set results with the lowest loss of 0.60738 and the best accuracy of 84.0%. The ResNet50 performed reasonably well, too. Even though the ResNet50 had about the same validation loss as the VGG16, its validation accuracy is 4.7% worse than the validation accuracy of VGG16. Also, the validation accuracy of ResNet50 is worse than the validation accuracy of MobileNet, regardless that the loss of ResNet50 is smaller. The VGG16 and MobileNet models achieved better validation accuracies than ResNet50. However, The ResNet50 might be more reliable in its predictions with the softmax output activation. This was discussed in Subsection 2.2.2. Small-CNN performed poorly compared to the other models. In the experiment, the architecture parameters of Small-CNN resulted in close to greatest values. Better results could have been achieved if randomization of a larger architecture was possible.

***Table 4.3.*** *Training results of the models.*

| Model | Train Loss | Val. Loss | Train Acc. | Val. Acc. |
|---|---|---|---|---|
| *Small-CNN* | 0.57754 | 0.87836 | 68.9% | 68.6% |
| *VGG16* | 0.10591 | **0.60738** | 94.0% | **84.0%** |
| *ResNet50* | 0.18134 | 0.61298 | 88.4% | 79.3% |
| *MobileNet* | **0.06977** | 0.69678 | **95.6%** | 80.1% |

The training implementation was evaluated. For this, the size of the model in parameters, training time, training epochs and average epoch time of the models were gathered to Table 4.4. Training of the ResNet50 lasted for the longest, for over nine hours. In contrast, ResNet50 was the only model which trained for the maximum 50 epochs. Training of the other models was stopped with early stopping policy. This had a great effect for overall training time. In order to compare the training of the models better, average time required for a training epoch with the models was calculated. With the larger models, average epoch lasted considerably longer. This is because the larger models have more parameters which are updated during training. With more parameters, more computation is required to update the weights of the network. After experimentation, the speed of training was not directly proportional to the size of the model. Even though the VGG16 was smaller than the ResNet50, average epoch of the VGG16 lasted for longer. The Small-CNN and MobileNet were faster to train the VGG16 and ResNet50 models.

In the experiments, training time of the models did not limit experimentation. However, if the dataset was considerably larger, experimentation could become more difficult. With

***Table 4.4.*** *Training details of the models.*

| Model | Params | Training Time | Epochs | Avg. Epoch |
|---|---|---|---|---|
| *Small-CNN* | 1.6M | 42min | 20 | 2.1min |
| *VGG16* | 21.1M | 190min | 12 | 15.8min |
| *ResNet50* | 24.6M | 550min | 50 | 11.0min |
| *MobileNet* | 3.5M | 149min | 27 | 5.5min |

larger datasets, for example ImageNet, training of a model could take up to days or even weeks [44]. If this was the case, for example, the hyperparameter optimization methods used in the experiments would become more difficult and thus more convenient approaches should be considered.

### 4.2.2 Classification Performance

The classification performance of the models was also analyzed by examining their class accuracies. The class accuracies of the models were gathered from the normalized confusion matrices (Figures 4.2, 4.4, 4.6 and 4.8) into Table 4.5. These accuracies were used to evaluate and compare the classification performance of each class in the dataset. Also, the average class accuracy (Avg. Acc.) metric was calculated by mean of the accuracies. The average class accuracy is used to evaluate classification with equal class importance.

***Table 4.5.*** *Class accuracies and average class accuracies of the models.*

| Model | Liv. being | Mach. | Nav. aid | Shore | Struc. | Vessel | Avg. Acc. |
|---|---|---|---|---|---|---|---|
| *Small-CNN* | 71% | 71% | 82% | 70% | 34% | 71% | 66.5% |
| *VGG16* | 78% | 76% | 80% | **88%** | 37% | **88%** | 74.5% |
| *ResNet50* | **87%** | **83%** | **87%** | 78% | **50%** | 85% | **78.3%** |
| *MobileNet* | 81% | 80% | 73% | 84% | 30% | 84% | 72.0% |

VGG16 classified Shore and Vessel classes both with the best accuracy of 88%. The ResNet50 classified the rest of the classes with the best results. When average class accuracy was calculated, ResNet50 achieved the best average class accuracy of 78.3%. The validation accuracies and the average class accuracies of the models were gathered into Table 4.6. It was observed, that the accuracies differed. Though the VGG16 achieved the best validation accuracy, it did not have the best average class accuracy. Because ResNet50 was able to classify the minority classes with higher accuracies, it was able to achieve the best average accuracy. It seems that the cost-sensitive approach has worked the best with ResNet50 as it was able to boost the accuracy of the minority classes.

Overall, the models performed reasonably well with Living being, Machine, Navigational aid, Shore and Vessel images. However, Structure images were classified correctly with poor accuracy. This was also noticed during the experiments when images of Structure were observed to be often misclassified as Shore. The causes for the misclassification were studied. For this, images of Structure were classified with using the MobileNet model. Six

***Table 4.6.*** *Validation accuracies and average class accuracies of the models.*

| Model | Val. Acc. | Avg. Acc. |
|---|---|---|
| *Small-CNN* | 68.6% | 66.5% |
| *VGG16* | **84.0%** | 74.5% |
| *ResNet50* | 79.3% | **78.3%** |
| *MobileNet* | 80.1% | 72.0% |

images of Structure, three correctly classified as Structure and three incorrectly classified as Shore, were selected to be examined. These images are shown in Figure 4.9. The correctly classified images were inspected. Two of the left-hand side images do not contain visible structures in them. The right-hand side image contains a house with a vessel in front of it. The incorrectly classified images were inspected. The left-hand side image clearly contains a structure with windows. The middle image contains shore-like features with a construction on the shoreline. The right-hand side of the image contains some kind of structure in it but no shore is seen. All of the images in Figure 4.9 look fairly similar. While some correctly classified images of Structure clearly have more Shore like features in them, some of the incorrectly ones have too. This similarity of these images be a source of misclassification because the images of these classes are not unambiguous.

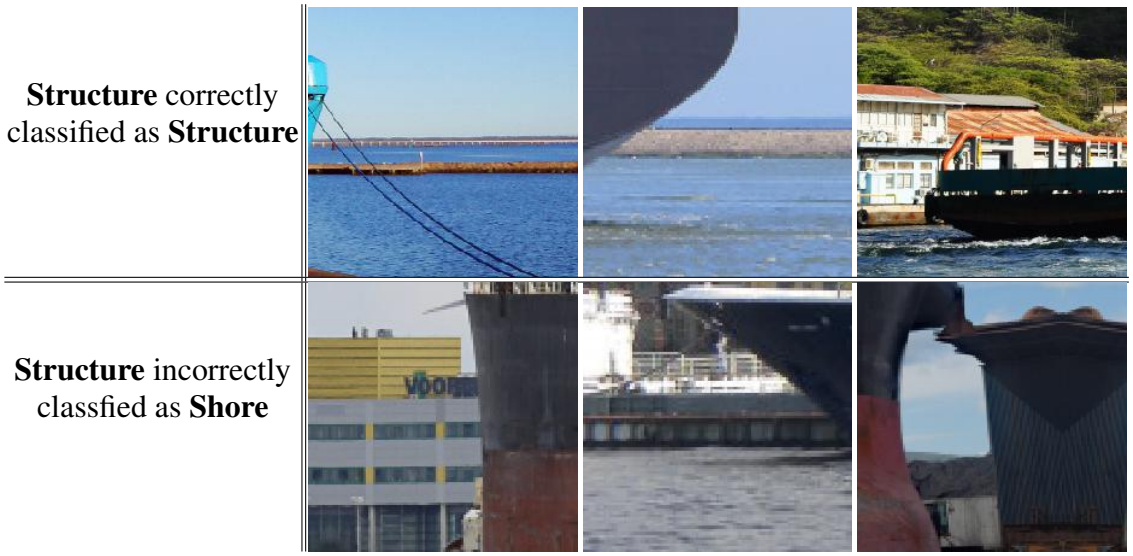



***Figure 4.9.*** *Images of Structure classified correctly as Structure and incorrectly as Shore with the MobileNet.*

Sources for the misclassification problem are analyzed. One possible source for the problem can be the preprocessing of the data. The original data contained the exact bounding box areas of the objects in the images. When the objects were extracted as single images the bounding boxes of the objects were expanded in order to retain the original aspect ratio when the image was resized. This may have added information of other objects into the images. Even though the data preprocessing methods may have added information which is not relevant for the object, it does not explain all sources for the confusion. For example, in Figure 4.9, some Structure images did not contain any Structure like features

in them. Because the dataset was not self-gathered, false labeling of images may be a possible source for the problem.

The RRMI dataset and its composition is discussed. What value does it add to have Structure class in the dataset? Clearly, when there is a structure it must be on shore to be there. Because the dataset was labeled and provided by Rolls-Royce, the full meaning of the class partition was unknown. However, the purpose of Structure class in the dataset was considered. The images of Structure could be merged with images of Shore. In this way, no relevant information about recognition would be lost and the classification task would become simple as fewer classes would be present. A naive approach was used to estimate the average class accuracy with the merging approach. In Table 4.7, average class accuracy is calculated with and without Structure class. ResNet50 would still have the best average accuracy with 84.0%. No further experimentation was performed with a modified dataset. In future studies, the RRMI dataset and its composition should be considered.

***Table 4.7.*** *Average accuracies of the models with and without Structure.*

| Model | Avg. Acc. with Structure | Avg. Acc. without Structure |
|---|---|---|
| *Small-CNN* | 66.5% | 73.0% |
| *VGG16* | 74.5% | 82.0% |
| *ResNet50* | **78.3%** | **84.0%** |
| *MobileNet* | 72.0% | 80.4% |

The classification performance of the ResNet50 was evaluated by plotting the ROC curves of each binary classification scenario using the output prediction of the model. For plotting, each prediction value was used as a threshold. The ROC curves and AUC scores are shown in Figure 4.10. The classifier performed excellently with Living being, Machine and Navigation. All these classes achieved an AUC score of 0.98. With these classes, the classifier could be tuned to achieve excellent classification performance. Vessel and Shore classes had AUC scores of 0.95 and 0.92, respectively. Structure class performed poorly with an AUC score of 0.81 as was expected after analyzing the class accuracies of the models. For the multi-label classification, a macro-average AUC score of 0.94 was achieved.

### 4.2.3 Computational Performance

In this subsection, computational performance is analyzed. The models were used to classify images with the CPU and the GPU in order to evaluate their real-time performance. In total, classification times for 100 images were measured. The results of these measurements are shown in Table 4.8 with the average time and 95% confidence interval. The rate of the GPU and CPU classification were compared. This is shown in the table as GPU/CPU ratio. The GPU required only a fraction of the time when compared with the CPU. The GPU times of the models were compared by generating a bar plot of the results, shown in Figure 4.11. The Small-CNN was the fastest model. The slowest model was

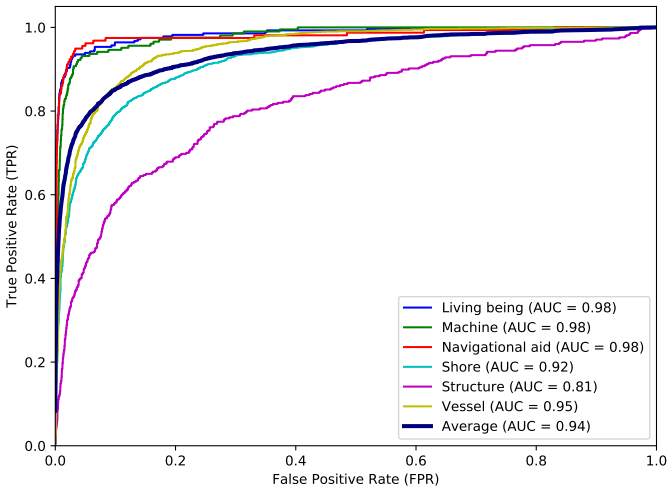



***Figure 4.10.*** *ResNet50 ROC curves and AUC scores.*

the ResNet50. The VGG16 and MobileNet models required only about half of the time compared to ResNet50. For implementation, the computational performance of the models should be evaluated in parallel with their accuracy. Nonetheless, time required for image classification with any of the models would not be an issue for a real-time application if a GPU was used.

***Table 4.8.*** *Image classification time with 95CI using CPU and GPU with the models.*

| Model | CPU (s) | GPU (s) | GPU/CPU |
|---|---|---|---|
| *Small-CNN* | $\mathbf{0.1166} \pm 0.0108$ | $\mathbf{0.0036} \pm 0.0009$ | 3.1% |
| *VGG16* | $1.1383 \pm 0.0163$ | $0.0086 \pm 0.0010$ | 0.8% |
| *ResNet50* | $0.5418 \pm 0.0149$ | $0.0160 \pm 0.0023$ | 3.0% |
| *MobileNet* | $0.1275 \pm 0.0047$ | $0.0073 \pm 0.0014$ | 5.7% |

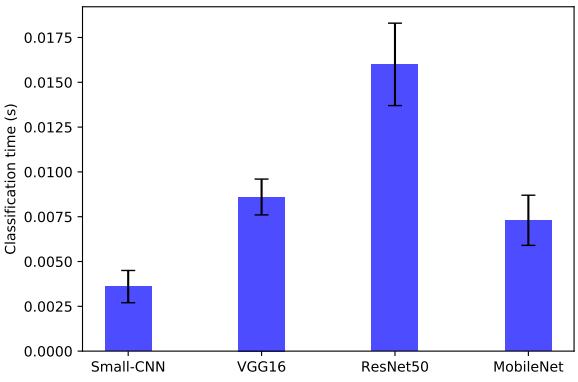



***Figure 4.11.*** *Image classification time using the GPU.*

# 5. DISCUSSION

This chapter provides general discussion on the results and research experimentation, compares them with related studies and considers their limitations. First, we evaluate the methods which were used for experimentation. After, we discuss on the results of the experiments. During the discussion, we also provide suggestions for future work.

## 5.1 Research Methods

In this section, the methods used for model and training implementation are discussed. First, we discuss on the model implementation approaches which were used in the experiments. Then, we consider possible limitations of the training parameters which were used in the training. Last, we discuss on the methods used for data preprocessing.

In the experiments, the models were trained with two distinct approaches with random weight initialization and transfer learning. The transfer learning experimentation was facilitated with Keras which allowed to use the architectures with the pretrained weights. Otherwise, the pretraining of the architectures could have taken even days [44]. Overall, Keras framework allowed an easy-to-use framework for neural network implementation.

First, Small-CNN architecture and its training parameters were randomized using random hyperparameter search. The approach allowed for an an easy way for experimentation. Because the process was automated, no time was required for manual hyperparameter tuning. However, the Small-CNN did not achieve good results for accuracy. In the experiments, it was observed that the hyperparameters that were randomized to generate the architecture of the network (See Table 4.1), were the maximum values except the kernel size which was one step from the maximum value. It seems that the training has favored a network of a bigger size. If more than 50 iterations were performed, it would have been possible that all the values would have been randomized to the maximum values. Because the training favored greater values, in future studies, randomization of network architecture hyperparameters could be done so that a larger network architecture would be possible. The Small-CNN was the smallest model of the four models with only 1.6 million parameters.

In addition, VGG16, ResNet50 and MobileNet architectures were pretrained with ImageNet data and trained to classify the dataset. Compared with the Small-CNN, which was trained random weight initialization, the models which were trained with transfer learning performed much better in classification with the RRMI dataset. The transfer learning approach proved to be a good method for training a classifier. In future studies, the benefits

of transfer learning could be studied more closely by using the same architectures by training them without pretraining and comparing the results with the pretrained models.

The methods and parameters used for training the models are discussed and their possible limitations are considered. First, we consider limitations of early stopping policy. Second, alternative approaches for learning rate tuning are discussed. Finally, we compare the data preprocessing methods with related studies.

In order to analyze early stopping, the training figures of the models (Small-CNN Figure 4.1, VGG16 Figure 4.3, ResNet50 Figure 4.5, MobileNet Figure 4.7) were examined. Early stopping served its purpose as no overfitting was observed. However, in some cases, early stopping may have been used too early. With Small-CNN and VGG16, the training was stopped after a few epochs. In addition, the validation accuracy of the Small-CNN varied greatly from epoch to epoch. In these cases, the good performance in the early epochs may have been achieved by chance. The training could have been continued for longer in order to reach better convergence. In this way, the results of these models would be more reliable. The loss of MobileNet converged to a steady level before early stopping. For Resnet50, early stopping criterion was not met.

Early stopping and its use in the experiments was evaluated by reviewing related studies. If training is well regularized, early stopping is not required at all, e.g. in [25][9]. A good classification accuracy may be achieved already with fewer than the maximum number of predefined epochs [25]. If the dataset and the network architecture are great, the training a DNN may require considerable time [44]. In these cases, if an adequate level of accuracy is achieved with fewer epochs, early stopping can save a considerable amount of time. In future studies, the use of early stopping requires extra consideration. Initial training iterations can be run to see if overfitting happens with a selected architecture and training parameters. If this happens, then the following iterations can be experimented with early stopping.

In the experiments with VGG16, ResNet50 and MobileNet, the models were trained by manually tuning the learning rate for each training iteration. It was observed that the learning rate had a great effect on the training performance. With a too large learning rate, training loss got stuck in the very beginning. With a too small learning rate, the training convergence was very slow. The manual tuning of the learning rate required a considerable amount of time as a new training iteration was required for every learning rate.

The training of a DNN can be performed with a dynamic learning rate where the training is initialized with an initial value which is dynamically reduced during the training [44][22]. In this way, time would not be wasted for extra iterations. In addition, dynamic learning rate can be used to reach a smaller loss when compared to static learning rate training [22]. With a static learning rate value, the loss of the network can plateau to a steady level during the training. If the value is dynamically adjusted in this plateau, the loss of the network can be further reduced in order to reach smaller loss [22]. In future studies, training with

dynamic learning rate could be used in order to quicken the training experimentation and to reach better results.

The RRMI dataset was constructed by extracting images of objects as single images. These single images varied greatly in resolution. Before training, the images were resized to a resolution that matched the input size of the used network. Both smaller than the target resolution and larger than the target resolution were used. The resize operation was performed using the nearest neighbour interpolation. The data preprocessing methods used in the experiments are evaluated and compared with related studies.

The methods used for data preprocessing in related studies differs. In some studies, images of objects were used only if the resolution of the image was the same or larger than the input resolution of the network [44][9][25]. In other studies, images of both smaller and larger resolution were used [51]. The methods used to image resizing differ, too. In some studies, the resizing is performed by retaining the aspect ratio[44][25]. In other studies, aspect ratio is not preserved [9][51]. In the research, there seems to be different ways for preprocessing and scaling the images. In the future, the effect of different methods could be studied.

## 5.2 Research Results

The rest of this chapter focuses on the research results. We discuss and reflect the results of experiments and compare them with related studies. First, we discuss on the classification performance of the models and consider their limitations. Then, the computational performance of the models is considered for a real-time application.

The best validation set accuracy of 84.0% for the RRMI dataset was achieved with VGG16. In [51] and [9] with maritime image datasets, accuracies of 81.9% and 80.1% were reported for the CNN models respectively. However, even as high as 98% accuracy has been achieved with deep CNNs models e.g. in [25]. It should be considered that the performance of a machine learning model is highly dependant on the dataset it was trained with. Because RRMI dataset was unique for the thesis experimentation, the absolute results can not be compared with related studies. If the accuracy was to be improved in the future, one of the best ways for this would be to gather more data [17, p.115].

The classification performance for the RRMI dataset was also evaluated by measuring class accuracies. The best average class accuracy of 78.3% was achieved with ResNet50. Even though a cost-sensitive approach was used, the accuracies and average class accuracies of the models varied. In addition, a systematic misclassification of Structure class was observed in the experiments. In future studies, the dataset and its composition should be considered. The dataset could be structured so that each class is equally represented so that a cost-sensitive training approach would not be required. In addition, the dataset could be studied for false labeling. Also, if images of Structure are not particularly important to be recognized they could be left out or merged with other images in the dataset in order to

improve classification performance. These issues with the dataset should be considered when the object recognition system is implemented in practice so that it serves its required purpose.

Because ResNet50 model had the highest average class accuracy, its classification performance was further analyzed by plotting the ROC curve of each class. With the validation set predictions, the model was able to distinguish Living being, Machine and Navigational aid very well at various classification thresholds. With images of Shore and Vessel, the model performed fairly. As expected, the model performed poorly with Structure. In the experiments, the classification was performed by selecting the output with the highest probability and no particular threshold was used for classification. In future studies, if classification was performed with ResNet50 using a threshold, the ROC curves could be used to determine the threshold value which resulted in the desired classifier performance for a selected class. In this way, the classifier could be tuned to achieve specific TPR and FPR for that class.

The computational performance of the models was not observed to be a limiting factor for real-time application. In the experiments, the models classified images between 3.6ms and 16.0ms when the GPU was used. The quickness of classification was observed to be similar with results in [9]. Time required for classification was also measured with the CPU and was observed to be much longer. It could be possible to improve CPU performance if acceleration methods were used. However, the main purpose of the experiments was to show that neural networks perform much faster using a GPU than a CPU.

Even with the slowest model, more than 60 images could be classified per second. It should be remembered, that these were the times required for object recognition. If the models were to be used in a system used for object detection, object localization and image extraction would be required too. These operations would require extra time. These factors were not studied in this research. The results of the experiments represent the times required for object recognition with the models using the software and hardware setup. For a real-time object detection system, the hardware, software and time required for every operation should be considered.

The purpose of the study was to study object recognition in order to implement a model to be used as a part of multi-sensor system for object detection with object localization and recognition. During the thesis work, the system was not implemented so the object recognition models were not tested in the field. For future work, the system could be implemented and the performance of the object recognition models could be tested in practice. Future work could also include reviewing other possible implementations for the system. Deep CNNs have been successfully applied to perform object detection with object localization and recognition with very good performance in multiple studies [15][38][23]. Though, when the localization is performed without a LiDAR sensor, the distance measurement would be lost. Nevertheless, object detection with CNNs would be an interesting topic for future work.

# 6. CONCLUSIONS

Machine learning can be used to solve various tasks by observing patterns in data. Classification is a machine learning task where an algorithm is trained with a dataset of input-output examples. In more complex classification tasks, such as object recognition, deep learning has shown promising results. In object recognition, deep CNNs have become the state of the art method.

Training a DNN for a task can be time-consuming. In order to train the DNN, a large amount of data is required. Without that much data, the training performance of the DNN can be improved with transfer learning where a pretrained model is used to improve the performance of a new model. Training performance of the DNN is also affected by various hyperparameters. Adjusting the hyperparameters in order to reach optimal training performance can be difficult. However, the process can be automated with practical approaches like grid and random hyperparameter search methods.

In the experiments, deep CNNs were trained to classify RRMI dataset images. For training and validation, a stratified 80/20 split of the dataset was used. Small-CNN architecture was generated and trained without pretraining using random hyperparameter search approach. VGG16, ResNet50 and MobileNet architectures were trained using transfer learning with weights which were pretrained with ImageNet data. Classification and computational performance of the models were measured in order to evaluate their potential for a real-time object recognition application.

The classification performance of the models was evaluated by measuring their accuracy and average class accuracy. The Small-CNN with random weight initialization performed poorly. Transfer learning, in turn, with VGG16, ResNet50 and MobileNet architectures proved to improve classification performance compared to Small-CNN. The VGG16 achieved the best accuracy of 84.0%. However, with average class accuracy, the ResNet50 achieved the best result of 78.3% as it was better in classifying classes with minor representation in the dataset. With the ResNet50, average AUC score of 0.94 was achieved. In the experiments, a systematic misclassification of Structure images in the dataset was observed. In future studies, the composition of the RRMI dataset should be considered in order to train a model with desired performance.

The computational performance was evaluated by measuring the time required for image classification with the models using a CPU and GPU. Image classification was much faster with the GPU, as expected. Time required for classification with the GPU spanned from 3.6 to 16.0 milliseconds. For a real-time application, time required for object recognition would not be a limiting factor. However, if the proposed object detection system was

implemented in practice the time required for other operations than image classification should be measured too.

Overall, the models performed well in classification with the RRMI dataset and were fast in computation. In the discussion, the research methods and results were discussed and compared with related studies. In related studies, better classification accuracies have been achieved using the same methods. In order to improve the classification performance, the used methods were discussed and evaluated. In future studies, early stopping would require extra consideration. Dynamic learning rate could be used for faster experimentation and improve the training performance. In addition, the effect of different data preprocessing methods for training performance could be studied.

Object recognition is a task in which deep learning methods have achieved excellent results. The thesis provided an overview on object recognition with deep CNNs and the models were found to be viable for real-time maritime application. In future studies, image classification performance could be improved by testing new methods. In addition, object detection with deep CNNs would provide an interesting topic for future work.

# REFERENCES


[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. Available: https://www.tensorflow.org/

[2] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, Journal of Machine Learning Research, Vol. 13, Iss. Feb, 2012, pp. 281–305.

[3] C.M. Bishop, Pattern recognition and machine learning, Springer, New York, 2006, 737 p.

[4] G. Bradski, A. Kaehler, Opencv, Dr. Dobb's journal of software tools, Vol. 3, 2000.

[5] M. Buda, A. Maki, M.A. Mazurowski, A systematic study of the class imbalance problem in convolutional neural networks, Neural Networks, Vol. 106, 2018, pp. 249–259.

[6] R. Caruana, S. Lawrence, C.L. Giles, Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping, in: Advances in neural information processing systems, 2001, pp. 402–408.

[7] H. Cho, Y.W. Seo, B.V. Kumar, R.R. Rajkumar, A multi-sensor fusion system for moving object detection and tracking in urban driving environments, in: Robotics and Automation (ICRA), 2014 IEEE International Conference on, IEEE, 2014, pp. 1836–1843.

[8] F. Chollet *et al.*, Keras, 2015. Available: https://keras.io

[9] C. Dao-Duc, H. Xiaohui, O. Morère, Maritime vessel images classification using deep convolutional neural networks, in: Proceedings of the Sixth International Symposium on Information and Communication Technology, ACM, 2015, pp. 276–281.

[10] DIMECC, D4value home page, 2018. Accessed: 1.1.2018. Available: http://d4value.dimecc.com/

[11] A. Eitel, J.T. Springenberg, L. Spinello, M. Riedmiller, W. Burgard, Multimodal deep learning for robust rgb-d object recognition, in: Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, IEEE, 2015, pp. 681–687.

[12] E. Fast, E. Horvitz, Long-term trends in the public perception of artificial intelligence, in: AAAI, 2017, pp. 963–969.

[13] T. Fawcett, An introduction to roc analysis, Pattern Recognition Letters, Vol. 27, Iss. 8, ROC Analysis in Pattern Recognition, 2006, pp. 861 – 874.

[14] G. Forman, M. Scholz, Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement, ACM SIGKDD Explorations Newsletter, Vol. 12, Iss. 1, 2010, pp. 49–57.

[15] R. Girshick, Fast r-cnn, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440–1448.

[16] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, 2011, pp. 315–323.

[17] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, 781 p. Available: http://www.deeplearningbook.org

[18] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, T. Chen, Recent advances in convolutional neural networks, Pattern Recognition, Vol. 77, 2018, pp. 354 – 377.

[19] D. Han, Q. Liu, W. Fan, A new image classification method using cnn transfer learning and web data augmentation, Expert Systems with Applications, Vol. 95, 2018, pp. 43 – 56.

[20] T. Hastie, R. Tibshirani, J. Friedman, The elements of statistical learning: data mining, inference and prediction, 2nd ed., Springer, New York, 2009, 744 p.

[21] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Transactions on Knowledge & Data Engineering, Iss. 9, 2008, pp. 1263–1284.

[22] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[23] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, CoRR, Vol. abs/1704.04861, 2017.

[24] J.D. Hunter, Matplotlib: A 2d graphics environment, Computing In Science & Engineering, Vol. 9, Iss. 3, 2007, pp. 90–95.

[25] H. Huttunen, F.S. Yancheshmeh, K. Chen, Car type recognition with deep neural networks, in: Intelligent Vehicles Symposium (IV), IEEE, 2016, pp. 1115–1120.

[26] IMO, International maritime organization. Accessed: 26.4.2018. Available: http://www.imo.org/en/Pages/Default.aspx

[27] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, 2015, JMLR.org, ICML'15, Lille, France, pp. 448–456.

[28] E. Jones, T. Oliphant, P. Peterson *et al.*, SciPy: Open source scientific tools for Python, 2001. Accessed: 1.8.2018. Available: http://www.scipy.org/

[29] S.B. Kotsiantis, I. Zaharakis, P. Pintelas, Supervised machine learning: A review of classification techniques, Emerging artificial intelligence applications in computer engineering, Vol. 160, 2007, pp. 3–24.

[30] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, 2012, pp. 1097–1105.

[31] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature, Vol. 521, Iss. 7553, 2015, pp. 436–444.

[32] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural computation, Vol. 1, Iss. 4, 1989, pp. 541–551.

[33] H. Liu, Pre-training in convolutional neural networks, master's thesis, Tampere University of Technology, Faculty of Computing and Electrical Engineering, Tampere, 2016, 68 p. Available: https://dspace.cc.tut.fi/dpub/handle/123456789/24458

[34] T.M. Mitchell, Machine learning, McGraw-Hill, New York, 1997, 414 p.

[35] G. Monteiro, C. Premebida, P. Peixoto, U. Nunes, Tracking and classification of dynamic obstacles using laser range finder and vision, in: Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2006, pp. 1–7.

[36] Oxford dictionary, https://en.oxforddictionaries.com/definition/artificial_intelligence. Accessed: 2018-08-01.

[37] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Transactions on knowledge and data engineering, Vol. 22, Iss. 10, 2010, pp. 1345–1359.

[38] S. Ren, K. He, R. Girshick, J. Sun, Faster r-cnn: Towards real-time object detection with region proposal networks, in: Advances in neural information processing systems, 2015, pp. 91–99.

[39] G. Rossum, Python Reference Manual, Techn. rep., Amsterdam, The Netherlands, 1995.

[40] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet Large Scale Visual Recognition Challenge, International Journal of Computer Vision (IJCV), Vol. 115, Iss. 3, 2015, pp. 211–252.

[41] S.J. Russell, P. Norvig, Artificial intelligence: a modern approach, 3rd ed., Pearson Education, Upper Saddle River, N.J.Harlow, 2010, 1132 p.

[42] A.L. Samuel, Some studies in machine learning using the game of checkers, IBM Journal of research and development, Vol. 3, Iss. 3, 1959, pp. 210–229.

[43] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Networks, Vol. 61, 2015, pp. 85 – 117.

[44] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: International Conference on Learning Representations, 2015.

[45] M. Sokolova, G. Lapalme, A systematic analysis of performance measures for classification tasks, Information Processing & Management, Vol. 45, Iss. 4, 2009, pp. 427 – 437.

[46] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research, Vol. 15, Iss. 1, 2014, pp. 1929–1958.

[47] O. Stenroos, Object detection from images using convolutional neural networks; kohteentunnistus kuvista konvoluutioneuroverkoilla, G2 Pro gradu, diplomityö, 2017-08-28. Available: http://urn.fi/URN:NBN:fi:aalto-201709046859

[48] S.v.d. Walt, S.C. Colbert, G. Varoquaux, The numpy array: A structure for efficient numerical computation, Computing in Science and Engg., Vol. 13, Iss. 2, Mar. 2011, pp. 22–30.

[49] D.R. Wilson, T.R. Martinez, The general inefficiency of batch training for gradient descent learning, Neural Networks, Vol. 16, Iss. 10, 2003, pp. 1429–1451.

[50] J. Yosinski, J. Clune, Y. Bengio, H. Lipson, How transferable are features in deep neural networks?, in: Advances in neural information processing systems, 2014, pp. 3320–3328.

[51] M.M. Zhang, J. Choi, K. Daniilidis, M.T. Wolf, C. Kanan, Vais: A dataset for recognizing maritime imagery in the visible and infrared spectrums, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2015, pp. 10–16.